

Lecture 7: Functional and OO Design Descriptions

Executable Units

- Functional
 - Procedures
 - Functions
 - Subroutines
- Modules
 - Collections of procedures
 - Interface procedures
- Object Oriented
 - Classes
 - Interfaces
 - Class methods
- Components
 - Collections of classes
 - Interface classes

Functional Design

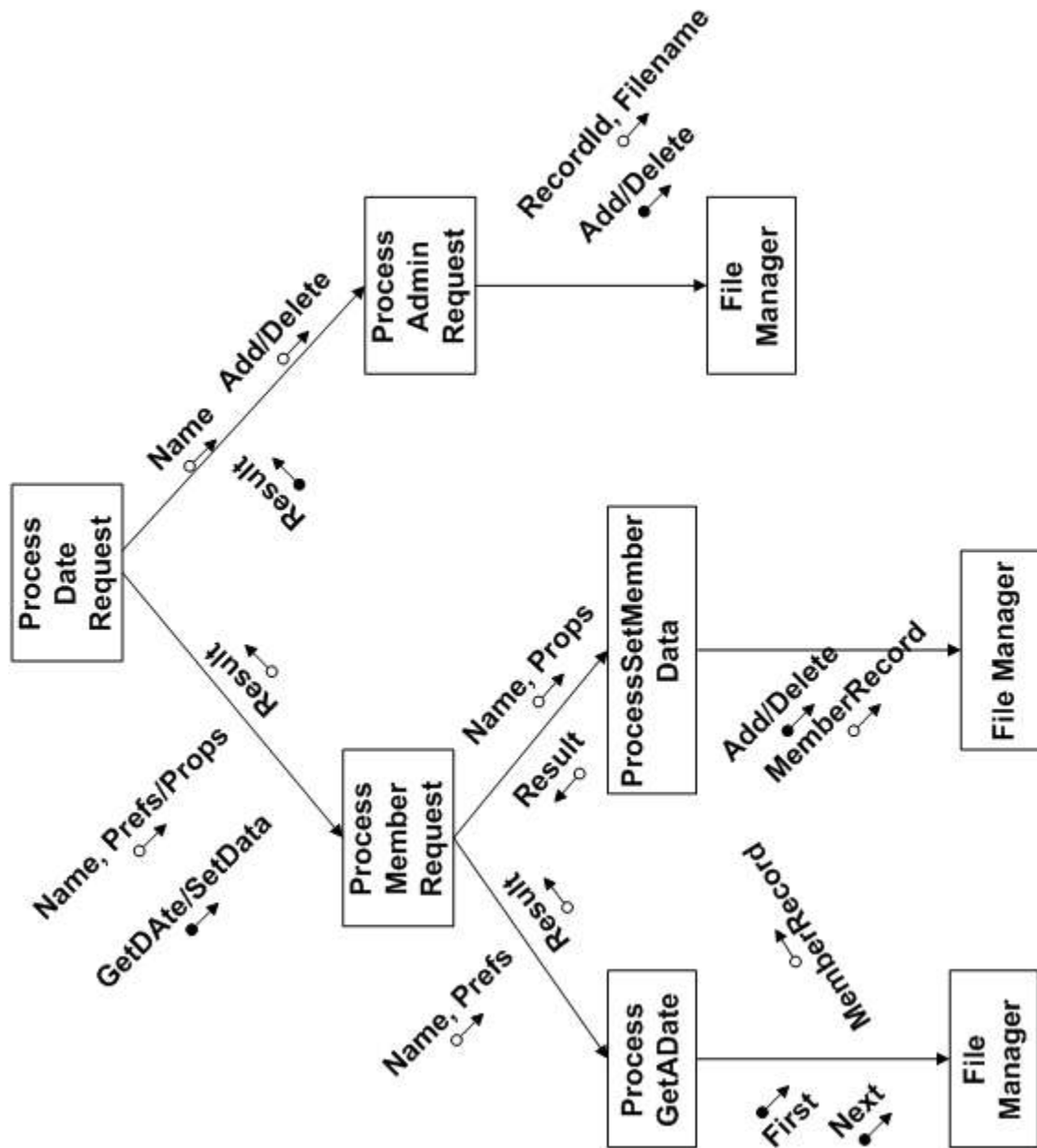
- Module connection/relationship models
- Design development
 - Top down design
 - Program refinement
- Module/Procedure call/use diagrams
- Algorithm design descriptions
 - Pseudo-Code
 - Nassi-Schneiderman diagrams
 - Decision Tables

Object Oriented Design

- Component connection/relationship models
- Design development
 - interaction sequence, collaboration diagrams
 - visibility analysis
- Design Diagrams – Class Structure
- Method design descriptions
 - algorithm design descriptions

Functional Design Diagrams – Function Interactions

- Tree structure
- Nodes are procedures or modules
- Arcs are calls or uses
- Arc notation: shows flow of data from one node to another: *data couples*
- Computational and control flow couples



DS Top Down Design

DS process

Dater or Admin process

Dater process

GetADate or SetMember Data process

Filemanager process

Admin process

Add or Delete process

FileManager process

DS PseudoCode and Program Refinement

```
Member Data getDate(DaterPreferences daterPrefs)
{
    Record record; Boolean match = false;
    record = filemanager("getFirst");
    see if record matches daterPrefs and set match
    while ((record != null) and (match == false))
    {
        record = filemanager("getNext");
        see if record matches daterPrefs and set match
    }
    if (match == false) return null else return record;
}
```

Data Visibility

For Functional Programs

- Data item: a function f uses or accesses a data item. How do we enable it to access and manipulate this data.
 - Local variable: declare one and assign data to it
 - Parameter: accessed passed data using formal parameter, or assign it to a local variable
 - Assign and reference data via a global variable

Type Visibility of Functional Programs

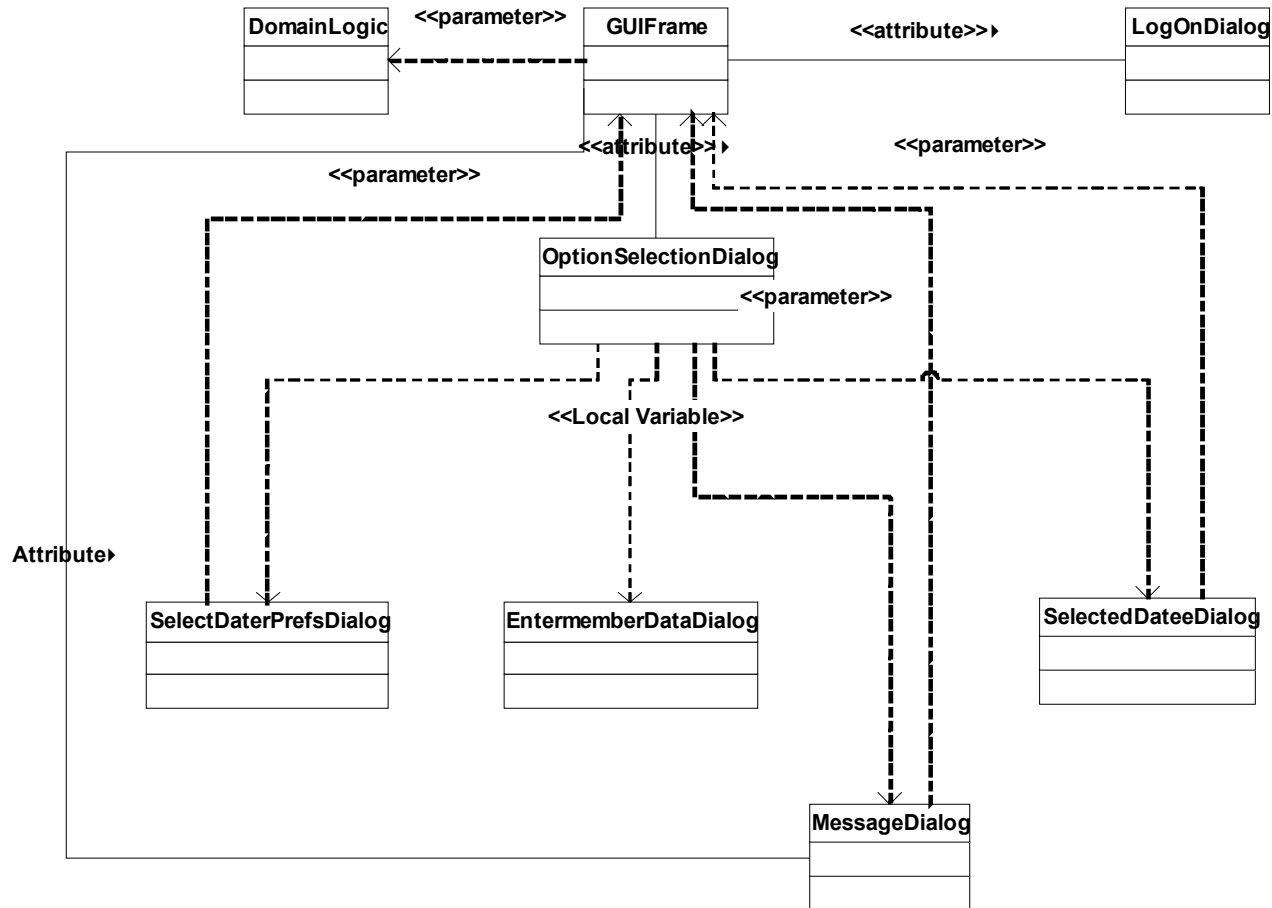
- Variable declarations involve a type
- Type definition visibility
 - Global generic types
 - Global struct definitions (C)
 - Local struct definitions
 - Includes files

Object Oriented Design Diagrams – Class Models

- Graph structure
- Nodes correspond to classes
- Arcs are relationships between classes
- Visibility analysis in class diagrams
 - Solid arc from A to B indicates A has a class variable whose values are instances of B
 - Dotted arcs indicate other kinds of visibility implementations

Diagram Details

- Nodes/Classes
 - Attributes: associated simple variables
 - Names and signatures of methods
 - Similar to Domain model classes, + methods
- Arcs
 - type of relationship, roles, multiplicity
- Separate details: for readability



Object Visibility for Object Oriented Designs

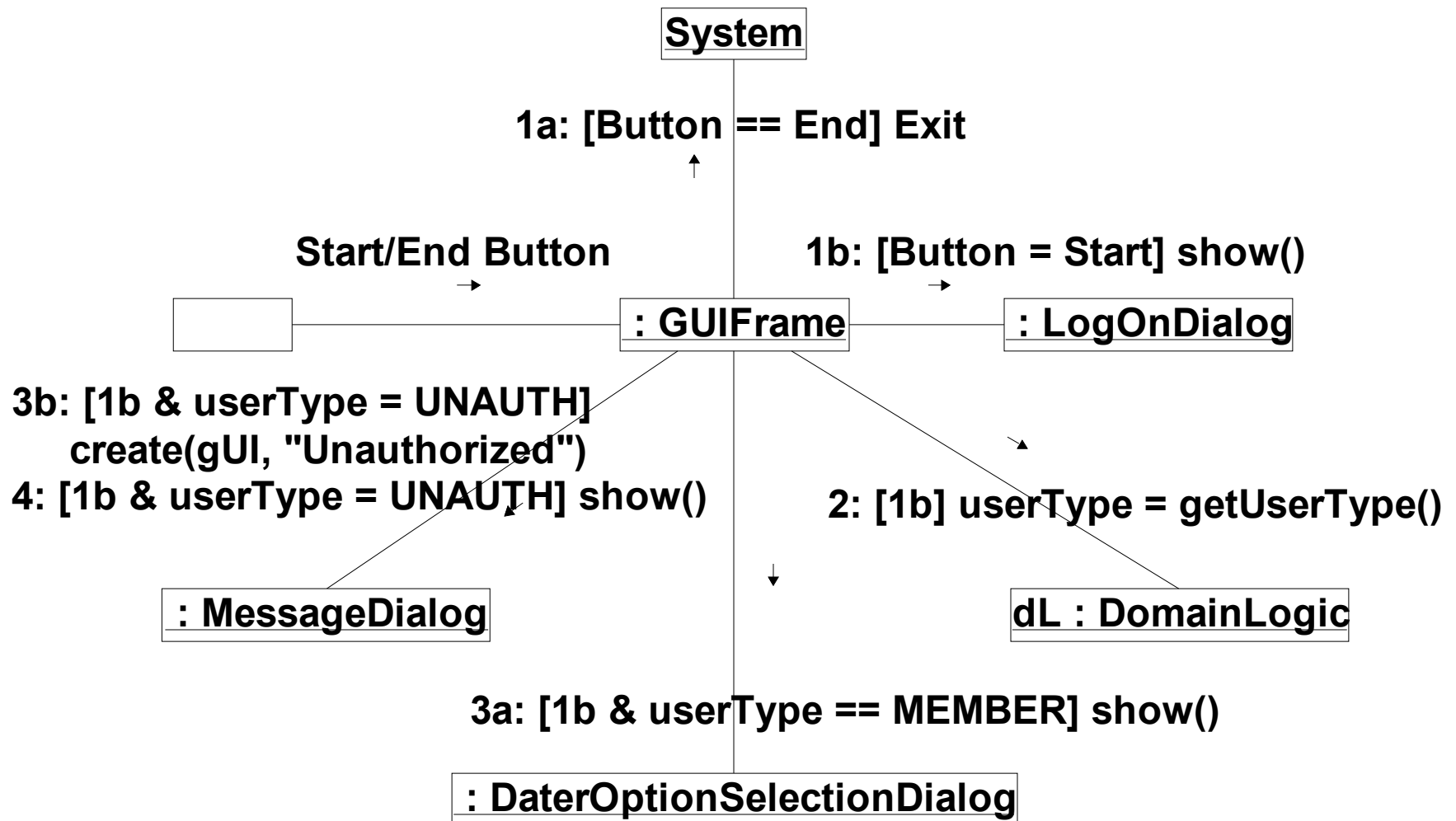
- An object X of class A has the ability to see or reference an object Y of class B
- E.g. in a collaboration diagram, X sends a message to Y
- Will require some way of achieving this visibility in the design
- Design Class Diagram can be used to document visibility implementation decisions

Ways to Achieve Object Visibility

- Attribute
- Parameter
- Local Variable
- Global

Attribute Visibility

- Class A can be defined with an attribute variable *y* whose values are of class B, which can be assigned the value Y
- E.g. :GUIFrame object in DS sends a `getUserType()` message to object `dL:DomainLogic`



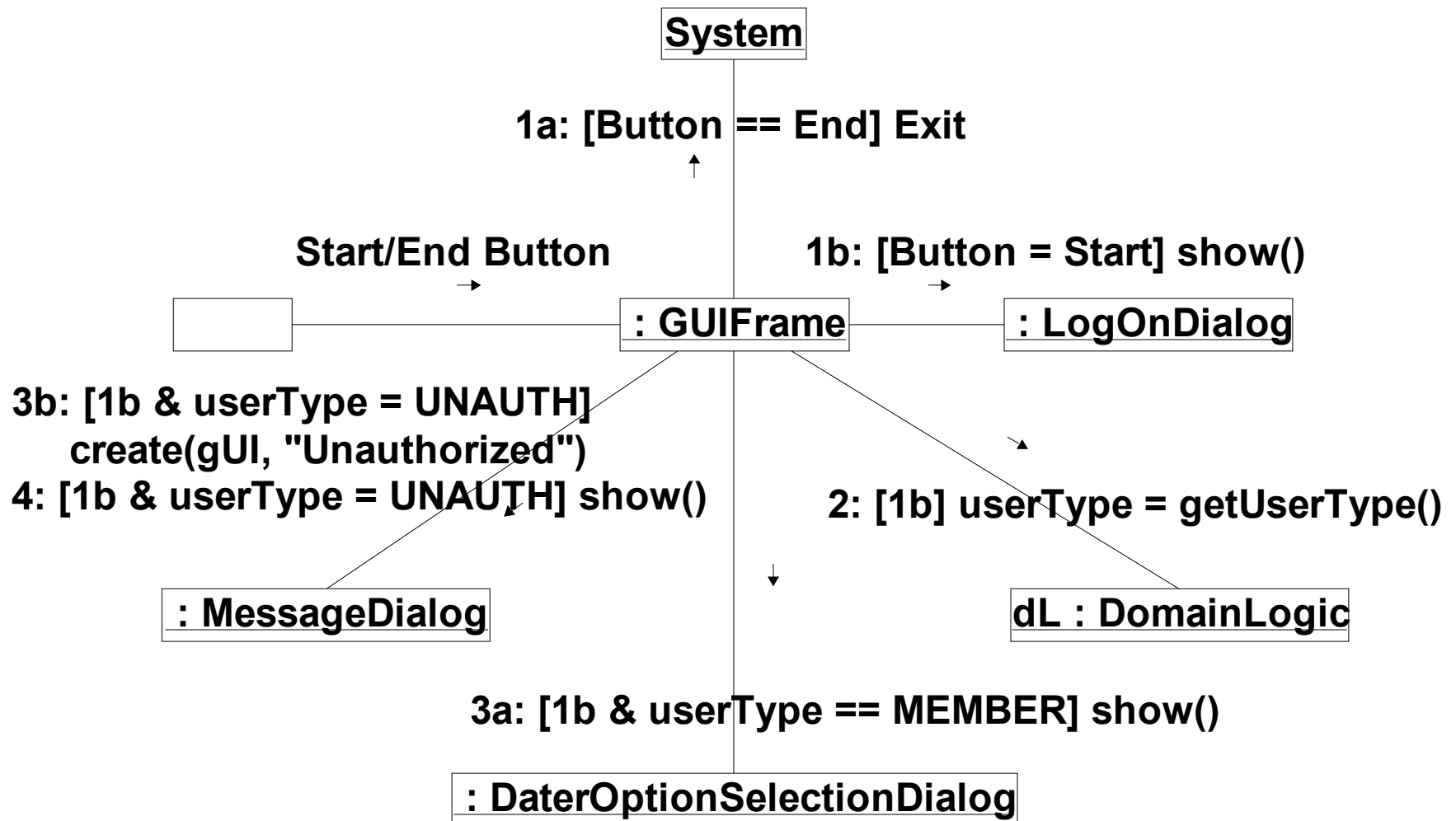
GUI Frame Presentation Logic

Attribute Visibility Solution

- Class GUIFrame, the class for the object : GUIFrame, will have an attribute/class variable dL of type DomainLogic whose value is the object dL.
- Permanent association needed. Attribute visibility is a good solution.

Parameter Visibility

- Y is referenced inside a method m() for X's class A. Y can be passed as a parameter y of m() which can be referenced in m()
- Eg. GUIFrame is going to set dL attribute variable to the dL Object, i.e. establish the attribute visibility in the earlier example.



GUI Frame Presentation Logic

Parameter Visibility Solution

- Parameter dL:DomainLogic will be passed as a parameter in the constructor for GUIFrame.
- Temporary parameter visibility is then changed to a more permanent attribute visibility

Local Variable Visibility

- A method `m()` in the class `A` for `X` can be defined with a local variable `y` of class `B`, that can be assigned to `Y`.
- E.g. In the GUI example, when a user selects the option of getting a date, the system will create and display an `SelectDataPrefsDialog`.

Local Variable Visibility Solution

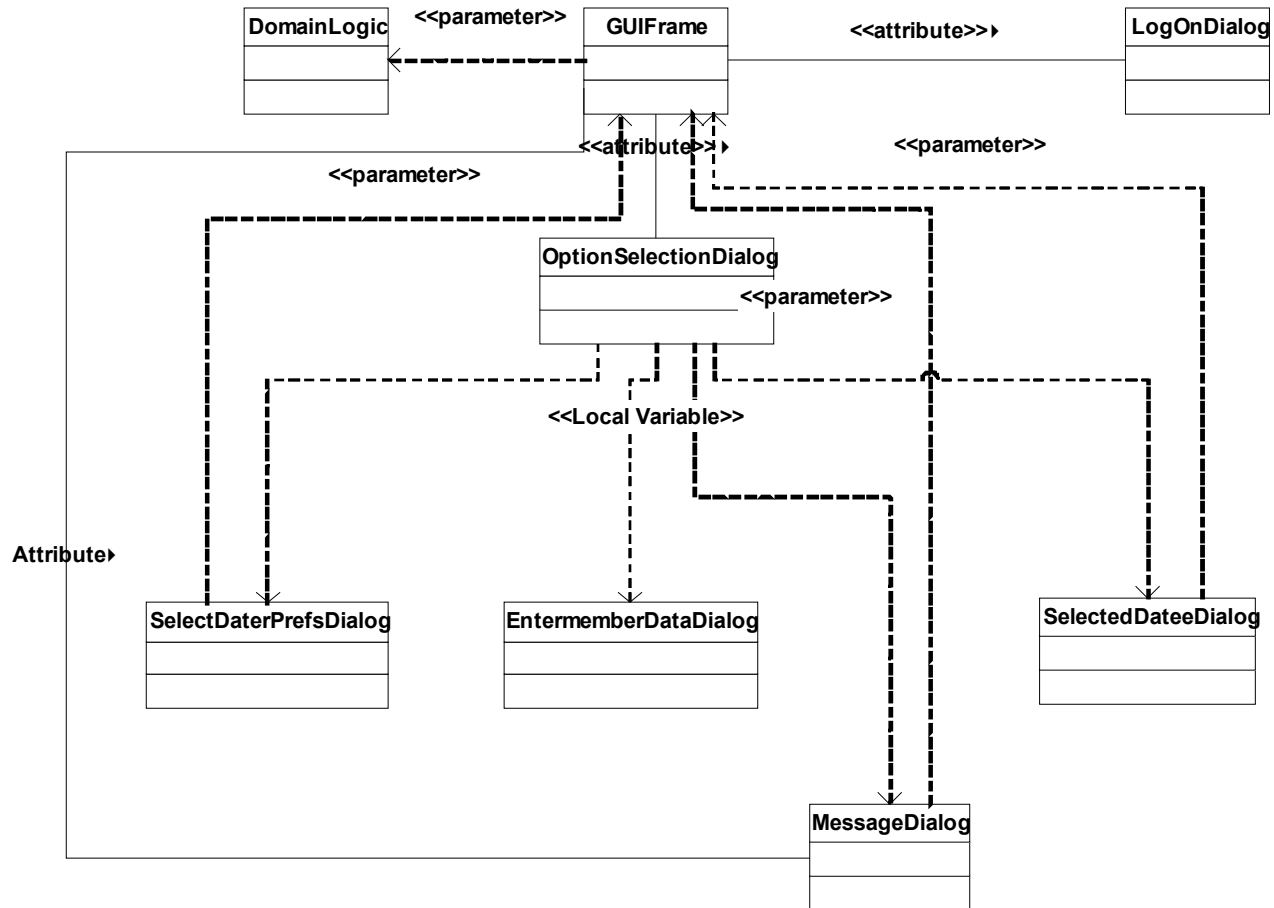
- Solution: the method in `DaterOptionsSelectionDialog` that is invoked when the `getDate()` button is pressed will have a local variable of type `SelectDaterPrefsDialog` that can be assigned to the object `:SelectDaterPrefsDialog` that is created at this point.

Global Visibility

- The object Y is made global to all classes so that all can reference it.
- E.g. The Message Dialog is used over and over in DS.
- Global visibility solution. Create a top level class that is global to all that use it. Assume it has a static object x whose value is dialog instance that can be re-used with different messages

GUI Example

- Simple diagram with relationships
- Sample detailed class descriptions given separately
 - Each has setVisible(), show(), a constructor and a method for responding to a button event that will be called by the OS



LogOnDialog
+OKButton: Button +nameField: TextField -parentFrame: Frame
+show() +setVisible() +LogOnDialog(): LogOnDialog() +buttonAction(in id:buttonID)

DaterOptionSelectionDialog
+getADate: Button +setMemberData: Button -parentFrame: Frame
+setVisible() +show() +buttonAction(in id : int) +DaterOptionSelectionDialog(in gUIGuiFrame)

O/O Programming and Class Visibility

- Declaration of an object of some class
 - E.g. Attribute/class variable, local variable (method), parameter (method), global
 - How is the class definition known in that context?
 - Determines structuring of class definitions
- Java: Inner class definition, file visibility, package visibility, import visibility

Class Visibility – Inner Classes

- Class A has an attribute variable y of class B
- Include the definition for class B inside class A
- E.g. in DS, all of the GUI class definitions are inner classes in GUI, which makes it possible to define instances of them

File Visibility

- A class in a file can access other class definitions in that file
- E.g. DS The class definition for the message class file is included in the GUI class file. GUI is the public object in this file (Java)

Package Visibility

- The classes in one file in a package can see the public and non-private classes in other files. Private classes in a file are not visible outside the file.

Import Visibility

- One file can declare itself to be part of a named package
- A file can import files in specified packages
- E.g. In DS GUI has parameter visibility to a Domain Logic object that is passed to its constructor. GUI has visibility to the DomainLogic class via an import statement

Domain Logic Examples

- Object visibility problem
 - An object X of class A has the ability to see or reference an object Y of class B
 - E.g. in a collaboration diagram, X sends a message to Y
 - Will require some way of achieving this visibility in the design

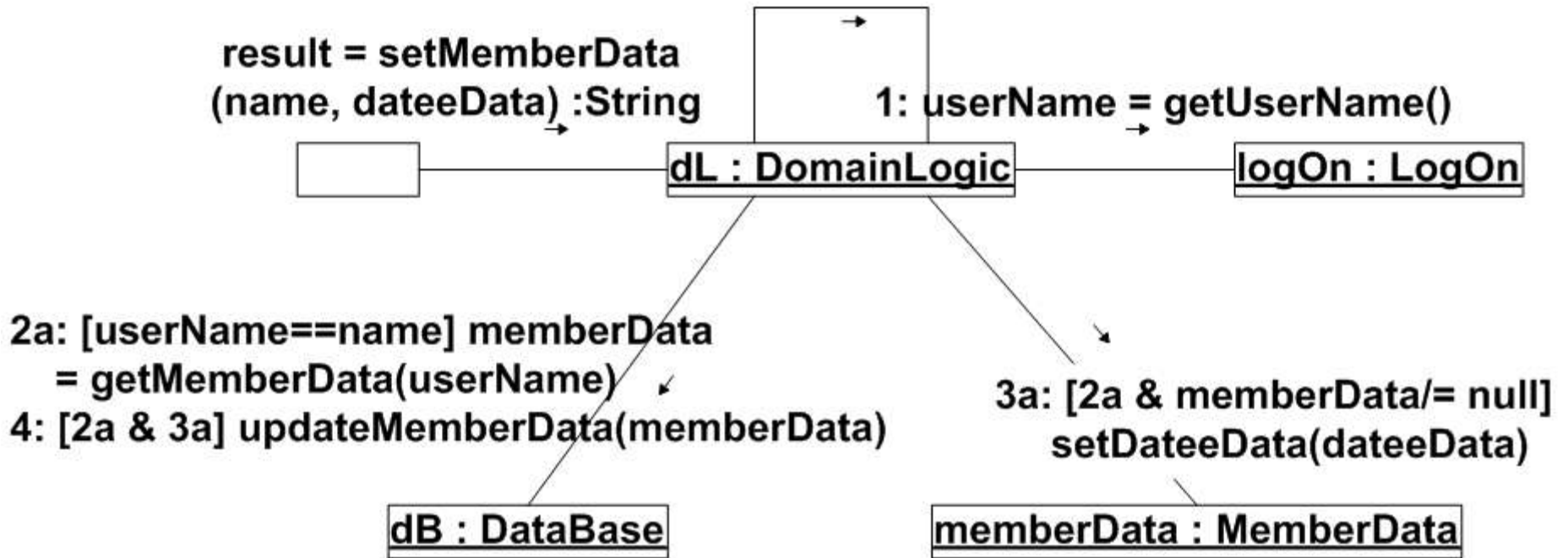
Object Visibility Solutions

- Attribute
- Parameter
- Local Variable
- Global

Attribute Visibility

- Class A can be defined with an attribute variable y whose values are of class B, which can be assigned the value Y
- E.g. dL:DomainLogic object in DS sends a `getMemberData()` message to object : DataBase

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

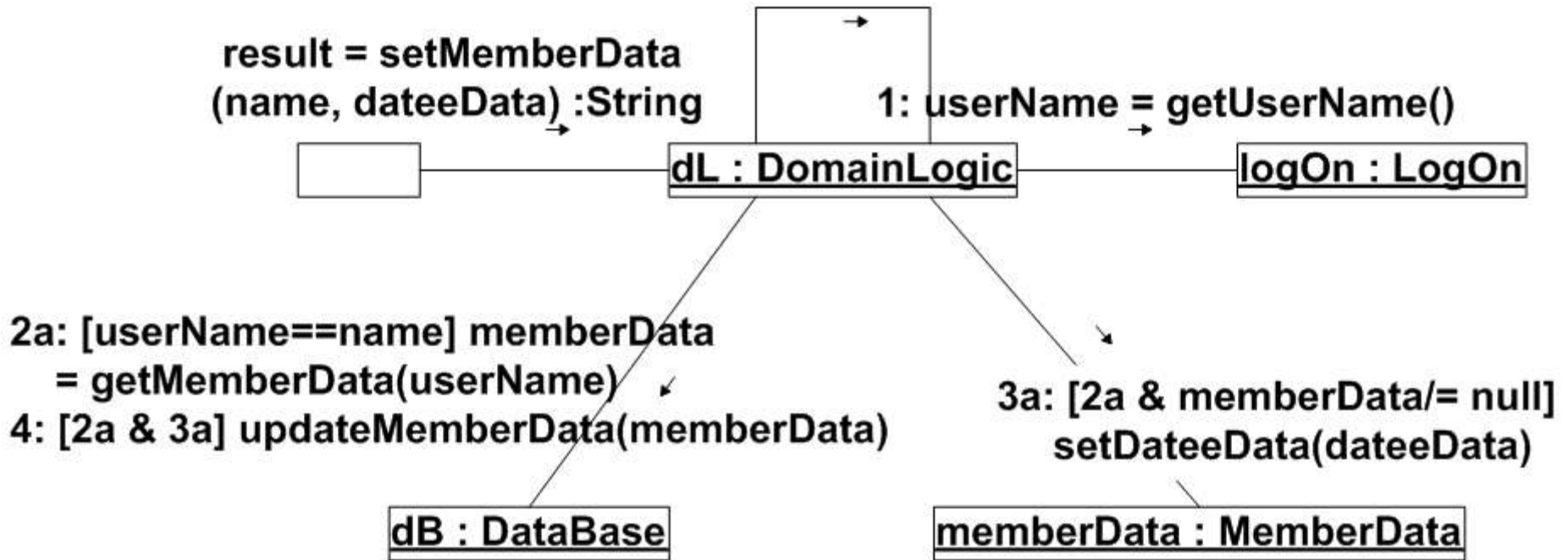
Attribute Visibility Solution

- Class DomainLogic, the class for the object dL:DomainLogic, will have an attribute/class variable dB of type DataBase whose value is the object dB.
- Permanent association needed. Attribute visibility is a good solution.

Parameter Visibility

- Y is referenced inside a method m() for X's class A. Y can be passed as a parameter y of M() which can be referenced in m()
- E.g. In the DS, DomainLogic is going to set dB attribute variable to the dB Object, i.e. establish the attribute visibility in the earlier example

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

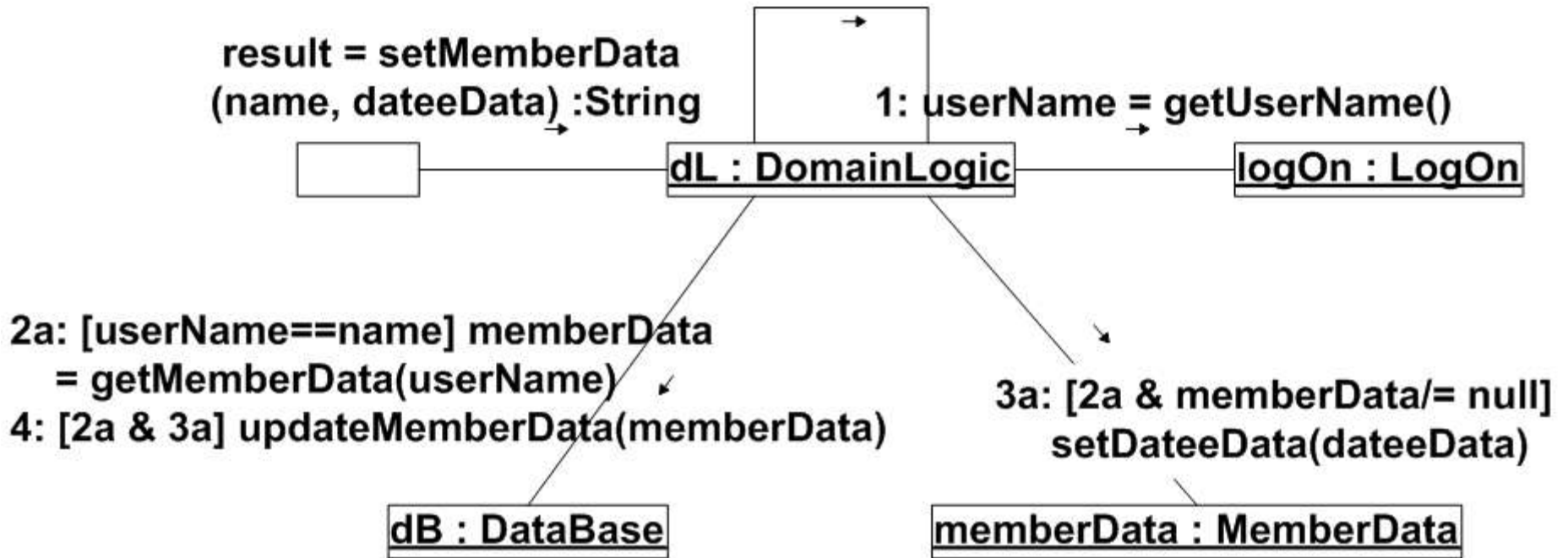
Parameter Visibility Solution

- Parameter dB:DataBase will be passed as a parameter in the constructor for DomainLogic.
- Temporary parameter visibility is then changed to a more permanent attribute visibility

Local Variable Visibility

- A method `m()` in the class `A` for `X` can be defined with a local variable `y` of class `B`, that can be assigned to `Y`.
- E.g. DS class `DomainLogic` has a method `setMemberData` that is called from GUI. It calls `getMemberData(userName)` in the data base to get back a member record which it then processes.

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

Local Variable Visibility Solution

- Solution: the `setMemberData()`, method will have a local variable `mD` which it can assign to the `MemberData` record returned from `getMemberData()`. This will enable it to maintain temporary visibility during the life of `setMemberData`.

Domain Logic Class Diagram

- Shows association relationships (attributes and other kinds of dependencies)
- Derived from above and other Collaboration Models
- Class details (attributes and methods) omitted
- Associations show class visibility also

