The Sequence Diagram

- Is used primarily to show the interactions between objects in the sequential order that those interactions occur.
- Not meant exclusively for developers
  - An organization's business staff can find sequence diagrams useful in communicating how the business currently works by showing how various business objects interact.
  - Besides documenting an organization's current affairs, a business-level sequence diagram can be used as a requirements document to communicate requirements for a future system implementation.

Sequence Diagram

- During the requirements phase of a project, analysts can take Use Cases to the next level by providing a more formal level of refinement. Then, Use Cases are refined into one or more sequence diagrams
- An organization's technical staff can find sequence diagrams useful in documenting how a future system should behave.
- During the design phase, architects and developers can use the diagram to force out the system's object interactions, thus fleshing out overall system design.

Sequence Diagram

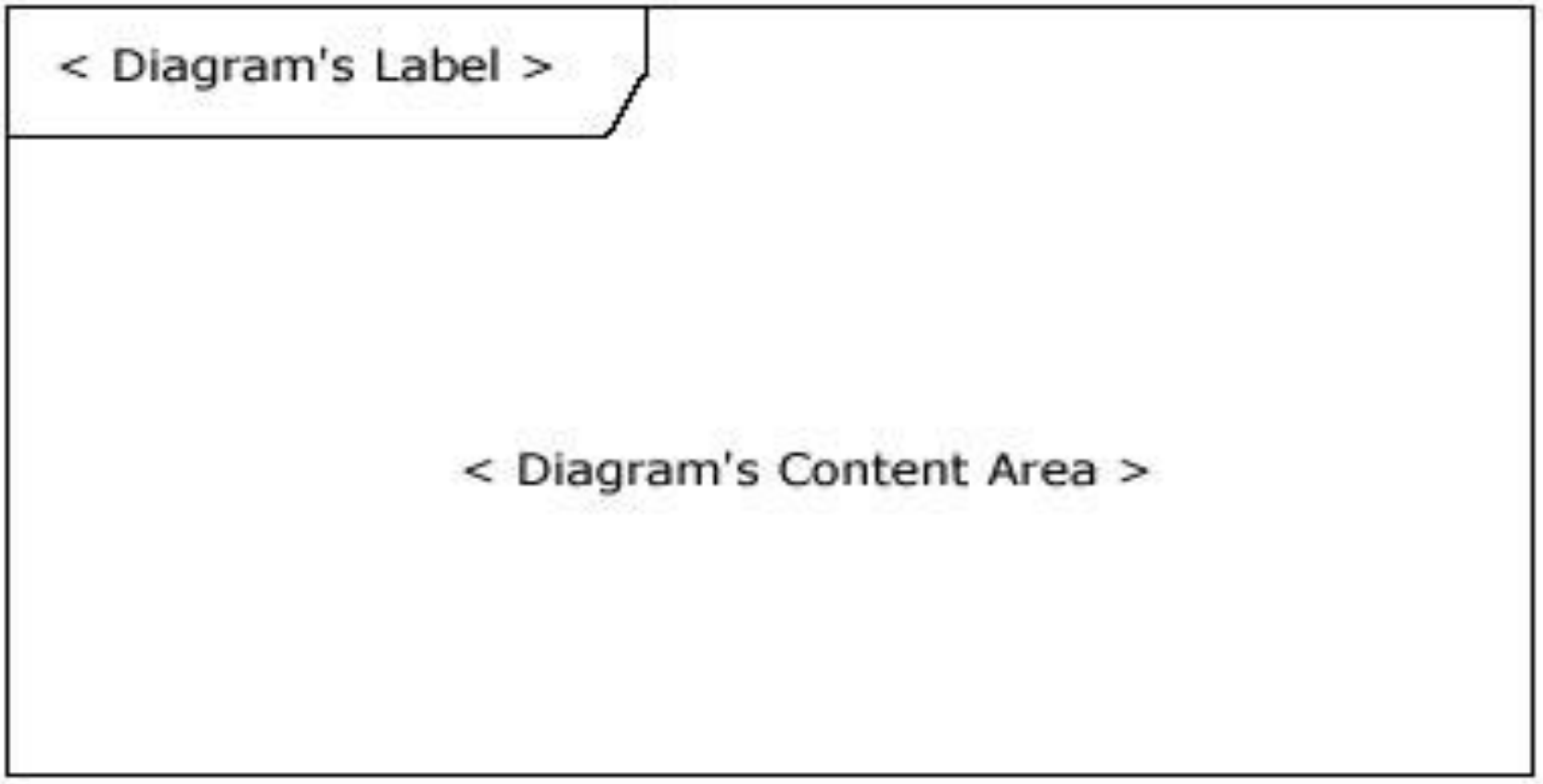
- One primary use of sequence diagrams is in the transition from requirements expressed as use cases to the next and more formal level of refinement.
- Use Cases are often refined into one or more sequence diagrams.
- In addition to their use in designing new systems, sequence diagrams can be used to document how objects in an existing system currently interact.

The Frame Element

- The frame element is used as a basis for many other diagram elements in UML 2; the first place most people will encounter a frame element is as the graphical boundary of a diagram.
- A frame element provides a consistent place for a diagram's label, while providing a graphical boundary for the diagram.

The Frame Element

Figure 1: An empty UML 2 frame element

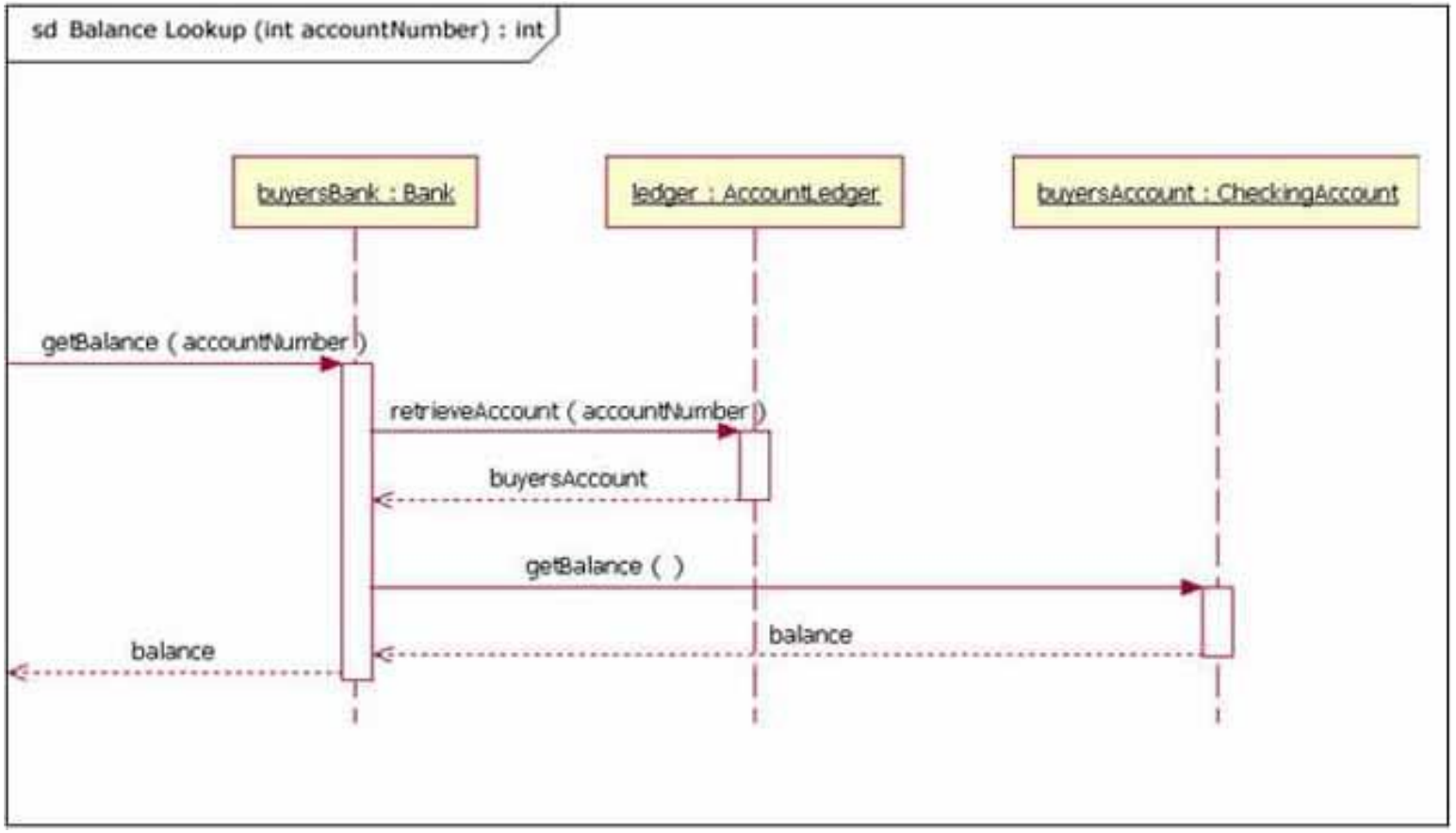


The Frame Element (2)

- In addition to providing a visual border, the frame element also depicts interactions, such as the sequence diagram.
  - On sequence diagrams incoming and outgoing messages (a.k.a. interactions) for a sequence can be modeled by connecting the messages to the border of the frame element (as seen in Figure 2).

Figure 2: Sequence Diagram

Diagram with incoming and outgoing messages



Frame Label

- Note the diagram's label begins with "sd" – for sequence diagram
- When using frame element to enclose diagram, the diagram's label follows the format

  Diagram Type Diagram Name

- UML provides specific text values for diagram types : Ex:
  - sd = Sequence Diagram
  - activity = Activity Diagram
  - use case = Use Case Diagram

Sequence Diagram Main Purpose

- Define event sequences that result in some desired outcome
  - Focus is less on messages themselves and more on the order in which messages occur
- Most sequence diagram communicates the messages sent between the system's objects as well as the order in which they occur
- The diagram conveys this information along the horizontal and vertical dimensions:

Conveying Information

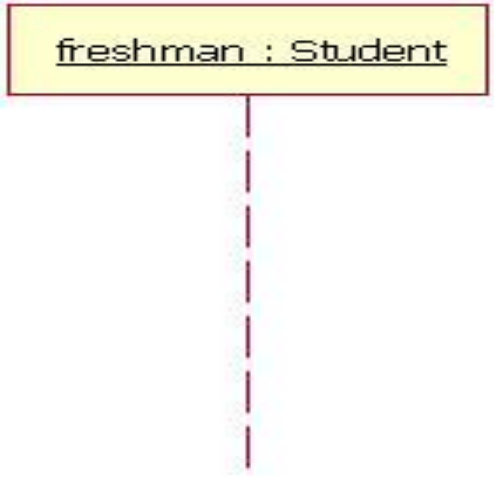
- Vertical dimension (top-down) shows the sequence of message calls as they occur
- Horizontal dimension (left to right) shows the object instances that the messages are sent to

Lifelines

- Lifeline notation elements are placed across the top of the diagram.
- Lifelines represent either roles or object instances that participate in the sequence being modeled.
  - Note: In fully modeled systems the objects (instances of classes) will also be modeled on a system's class diagram.
- Lifelines are drawn as a box with a dashed line descending from the center of the bottom edge (Figure 3).
- The lifeline's name is placed inside the box.

Figure 3: Lifeline

- Example of the Student class used in a lifeline whose instance name is Freshman



- UML standard for naming a lifeline:

  Instancename Name : Class Name

Figure 3 Lifeline

- Represents an instance of the class Student, whose instance name is freshman.
  - Note that, here, the lifeline name is underlined. When an underline is used, it means that the lifeline represents a specific instance of a class in a sequence diagram, and not a particular kind of instance (i.e., a role).
- In a future lecture we'll look at structure modeling. For now, just observe that sequence diagrams may include roles (such as *buyer* and *seller*) without specifying who plays those roles (such as **Pedro** and **Eugenia**). This allows diagram reuse in different contexts.
  - Simply put, instance names in sequence diagrams are underlined; roles names are not.

Lifeline Objects

- The example lifeline in Figure 3 is a named object, but not all lifelines represent named objects.
- Instead, a lifeline can be used to represent an anonymous or unnamed instance.
- When modeling an unnamed instance on a sequence diagram, the lifeline's name follows the same pattern as a named instance; but instead of providing an instance name, that portion of the lifeline's name is left blank.
- Again referring to Figure 3, if the lifeline is representing an anonymous instance of the Student class, the lifeline would be: “Student."
- Because sequence diagrams are used during the design phase of projects, it is completely legitimate to have an object whose type is unspecified: for example, "freshman."

Messages

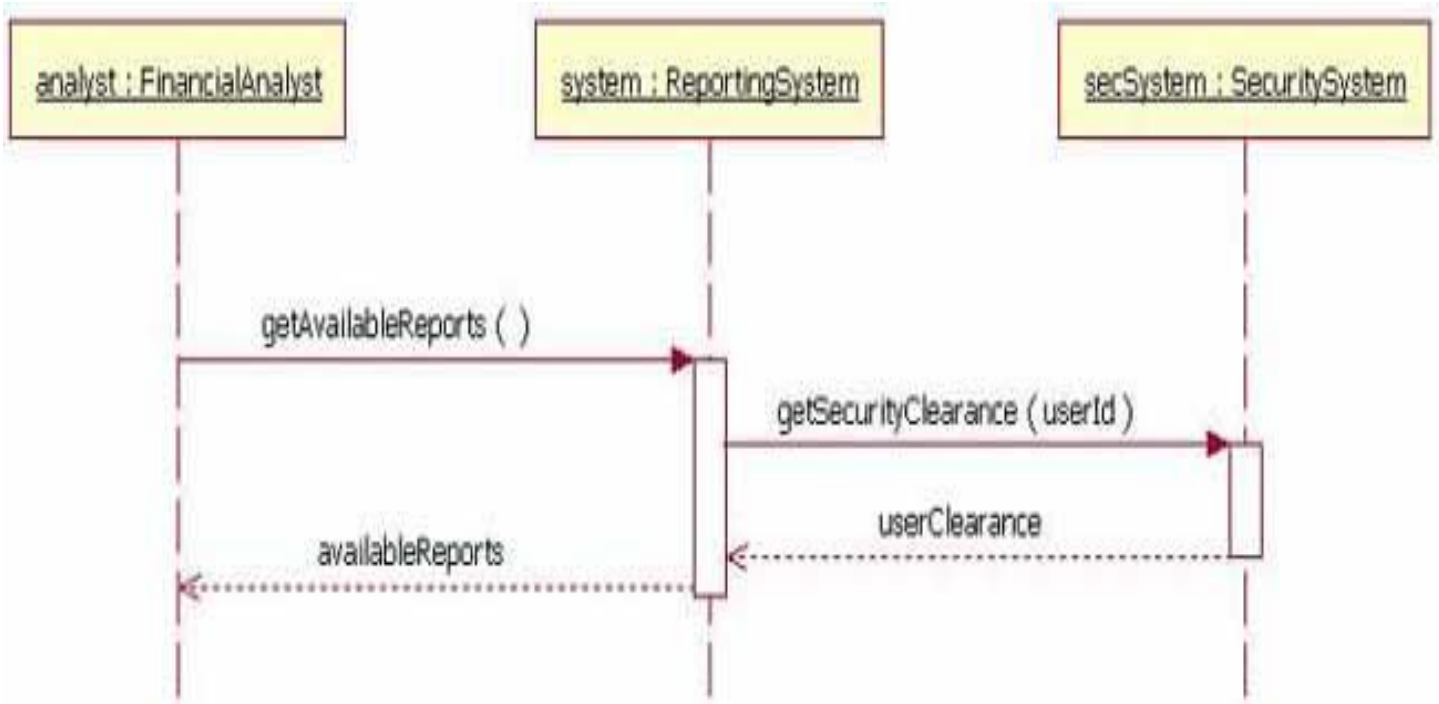
- The first message of a sequence diagram always starts at the top and is typically located on the left side of the diagram for readability.
  - Subsequent messages are then added to the diagram slightly lower than the previous message.
- To show an object (i.e., lifeline) sending a message to another object, draw a line to the receiving object with a solid arrowhead (if a synchronous call operation) or with a stick arrowhead (if an asynchronous signal).
- The message/method name is placed above the arrowed line.

Messages (2)

- The message that is being sent to the receiving object represents an operation/method that the receiving object's class implements.
- In the example in Figure 4, the analyst object makes a call to the system object which is an instance of the ReportingSystem class.
  - The analyst object is calling the system object's getAvailableReports method.
  - The system object then calls the getSecurityClearance method with the argument of userId on the secSystem object, which is of the class type SecuritySystem.
  - Note: When reading this sequence diagram, assume that the analyst has already logged into the system.

Messages (2)

Figure 4: Example of messages being sent between objects



Return Messages

- Besides just showing message-calls on the sequence diagram, Figure 4 diagram includes return messages.
- Although optional; - a return message is drawn as a dotted line with an open arrowhead back to the originating lifeline
  - Above this dotted line you place the return value from the operation.
  - In Figure 4 the secSystem object returns userClearance to the system object when the getSecurityClearance method is called. The system object returns availableReports when the getAvailableReports method is called.

Return Messages (2)

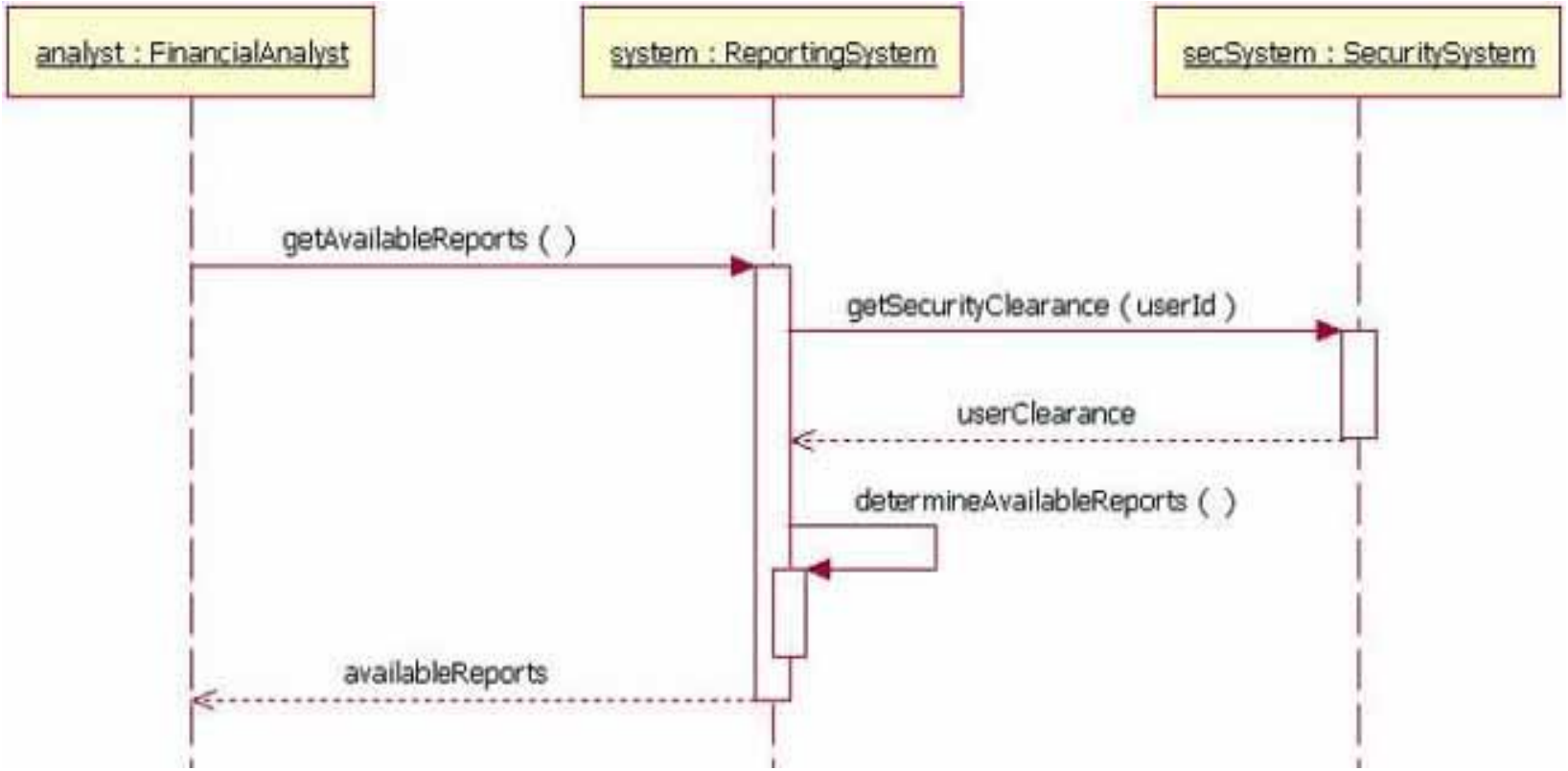
- Again, although optional, the use of return messages depends on the level of detail/abstraction that is being modeled.
- Return messages are useful if finer detail is required.
- Personally, I like to (and recommend) include return messages whenever a value will be returned, because the extra details make a sequence diagram easier to read.

Modeling

- When modeling a sequence diagram, there will be times that an object will need to send a message to itself.
- When does an object call itself?
- Modeling an object sending a message to itself can be useful in some cases.
  - For example, Figure 5 is an improved version of Figure 4. The Figure 5 version shows the system object calling its determineAvailableReports method.
- By showing the system sending itself the message "determineAvailableReports," the model draws attention to the fact that this processing takes place in the system object.

System Object Calling Itself

Figure 5: The system object calling its determineAvailableReports method

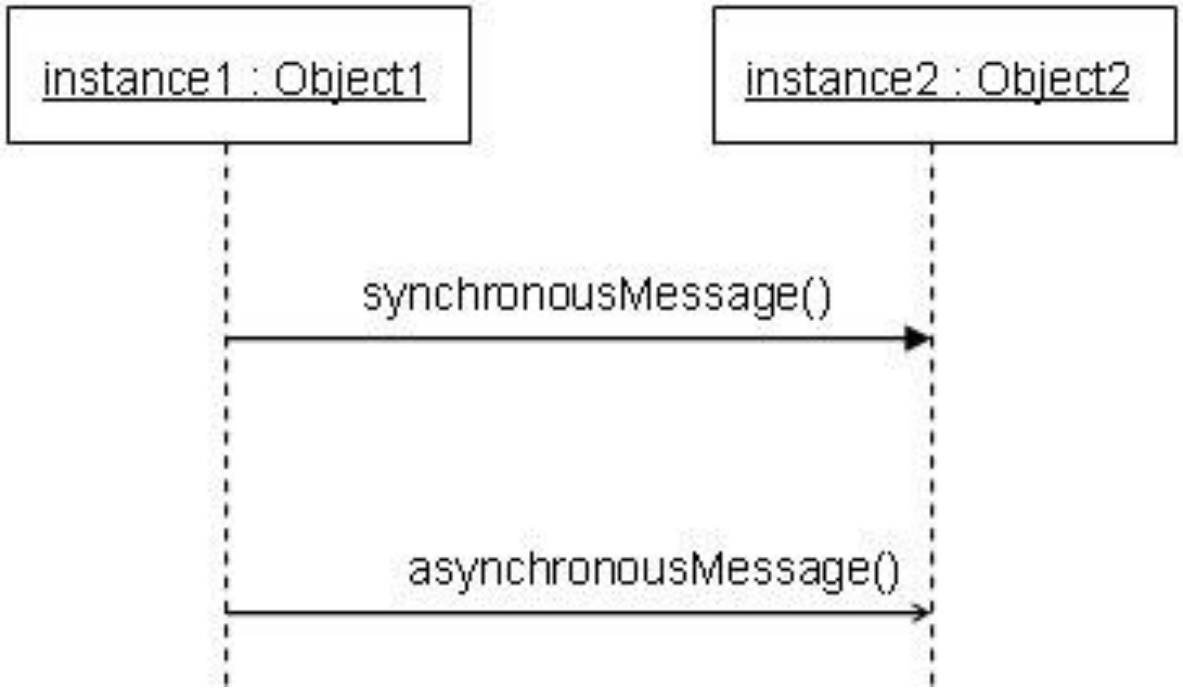


Object Calling itself

- To draw an object calling itself, draw a message as you would normally, but instead of connecting it to another object, you connect the message back to the object itself.
- Figure 5 show synchronous messages; however, in sequence diagrams you can also model asynchronous messages.
- An asynchronous message is drawn similar to a synchronous one, but the message's line is drawn with a stick arrowhead, as shown in Figure 6.

Asynchronous Message

Figure 6: A sequence diagram fragment showing an asynchronous message being sent to instance2

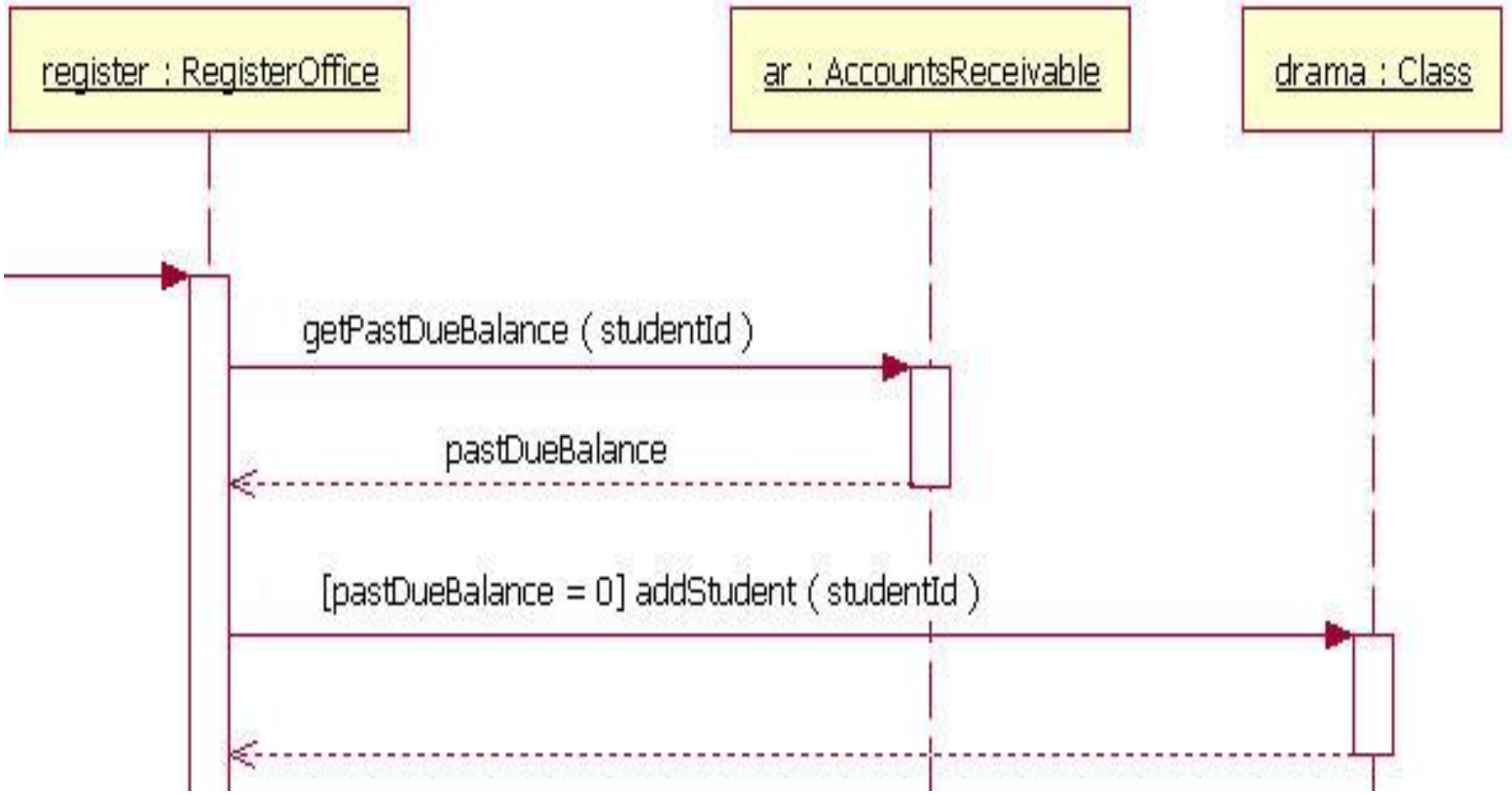


Guards

- When modeling object interactions, there will be times when a condition must be met for a message to be sent to the object.
- Guards are used throughout UML diagrams to control flow.
- To draw a guard on a sequence diagram in UML 1.x, you place the guard element above the message line being guarded and in front of the message name. Figure 7 shows a fragment of a sequence diagram with a guard on the message addStudent method.

Guards in UML 1.x

Figure 7: A segment of a UML 1.x sequence diagram in which the addStudent message has a guard



Guards (2)

- In Figure 7, the guard is the text

  "[pastDueBalance = 0]."

- By having the guard on this message, the addStudent message will only be sent if the accounts receivable system returns a past due balance of zero.
- The notation of a guard is very simple; the format is:

  [Boolean Test]

- For example,

  [pastDueBalance = 0]

Combined fragments (alternatives, options, and loops)

- In most sequence diagrams, however, the UML 1.x "in-line" guard is not sufficient to handle the logic required for a sequence being modeled.
- This lack of functionality was a problem in UML 1.x. UML 2 has addressed this problem by removing the "in-line" guard and adding a notation element called a Combined Fragment.
- A combined fragment is used to group sets of messages together to show conditional flow in a sequence diagram.
- The UML 2 specification identifies 11 interaction types for combined fragments.
- We touch on three of the eleven in this class "

Alternatives

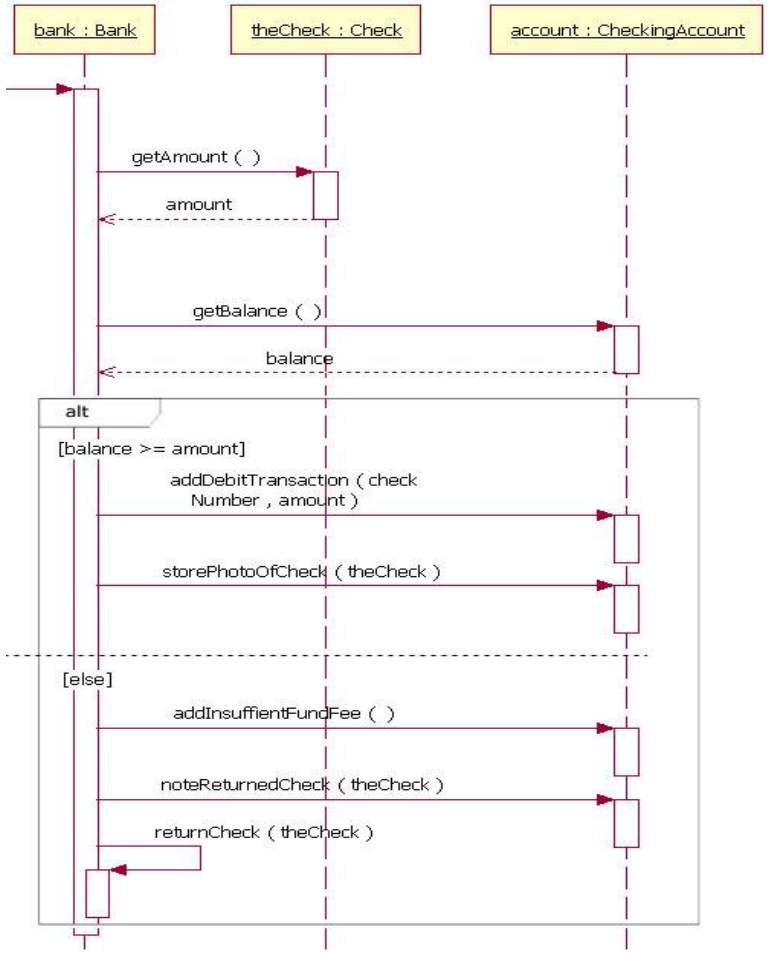
- Alternatives are used to designate a mutually exclusive choice between two or more message sequences.
  - [Note: It is possible for two or more guard conditions attached to different alternative operands to be true at the same time, but at most only one operand will actually occur at run-time (which alternative "wins" in such cases is not defined by the UML standard).]
- Alternatives allow the modeling of the classic "if then else" logic (e.g., **if** I buy three items, **then** I get 20% off my purchase; **else** I get 10% off my purchase).

Alternatives (2)

- In Figure 8, an alternative combination fragment element is drawn using a frame.
- The word "alt" is placed inside the frame's namebox.
  - The larger rectangle is then divided into what UML 2 calls operands.
  - [Note: Although operands look a lot like lanes on a highway, I specifically did not call them lanes. Swim lanes are a UML notation used on activity diagrams.
- Operands are separated by a dashed line. Each operand is given a guard to test against, and this guard is placed towards the top left section of the operand on top of a lifeline.

Alternatives (3)

Figure 8: A sequence diagram fragment that contains an alternative combination fragment



Example of Alternative Combination Fragment

- Figure 8 shows the sequence starting at the top
  - with the bank object getting the check's amount and the account's balance.
  - At this point in the sequence the alternative combination fragment takes over.
    - Because of the guard "[balance >= amount]," if the account's balance is greater than or equal to the amount, then the sequence continues with the bank object sending the addDebitTransaction and storePhotoOfCheck messages to the account object.
    - However

Fragment contd.

- if the balance is not greater than or equal to the amount, then the sequence proceeds with the bank object sending the addInsuffientFundFee and noteReturnedCheck message to the account object and the returnCheck message to itself.

- The second sequence is called when the balance is not greater than or equal to the amount because of the "[else]" guard.
- In alternative combination fragments, the "[else]" guard is not required; and if an operand does not have an explicit guard on it, then the "[else]" guard is to be assumed.