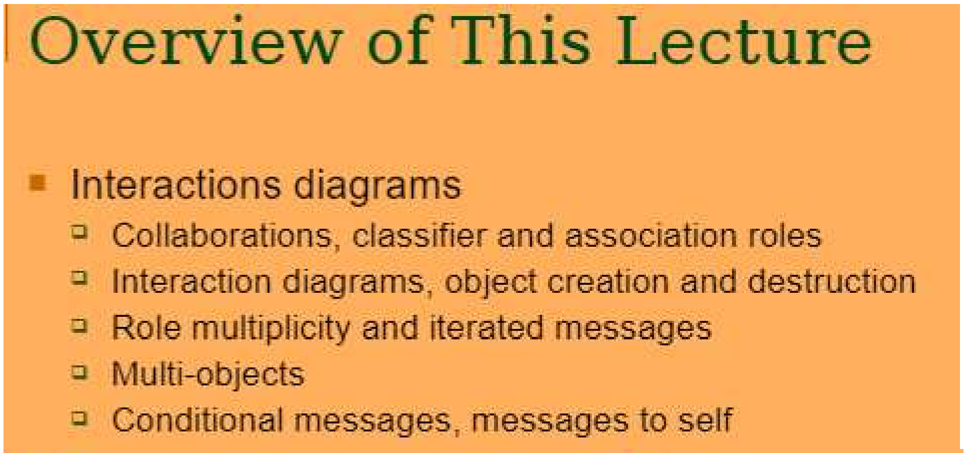
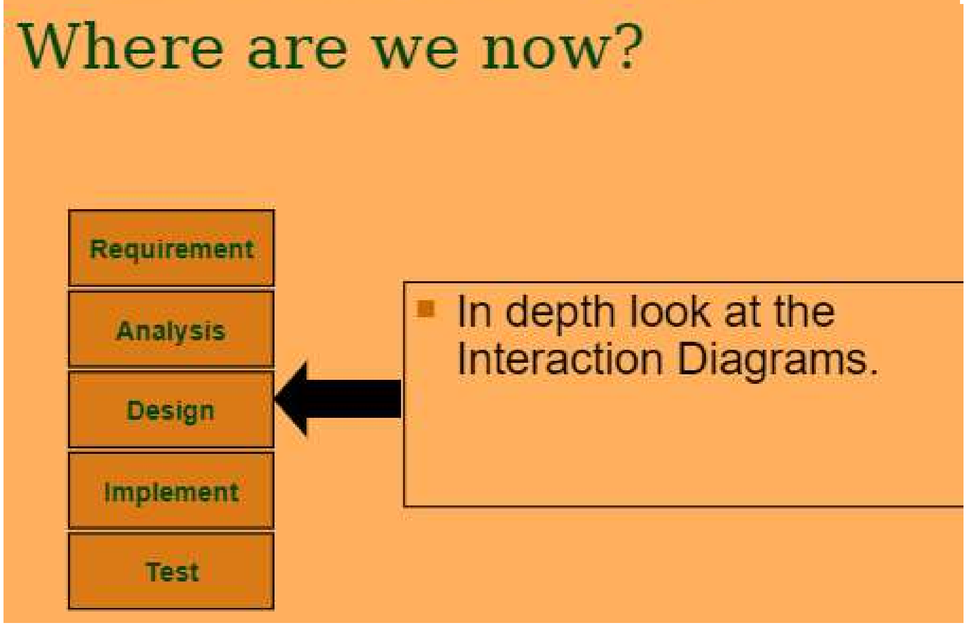
# Lecture 7

Monday, February 27, 2023 9:43 AM

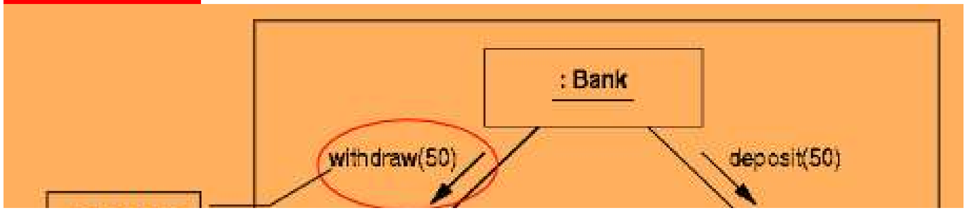
- 

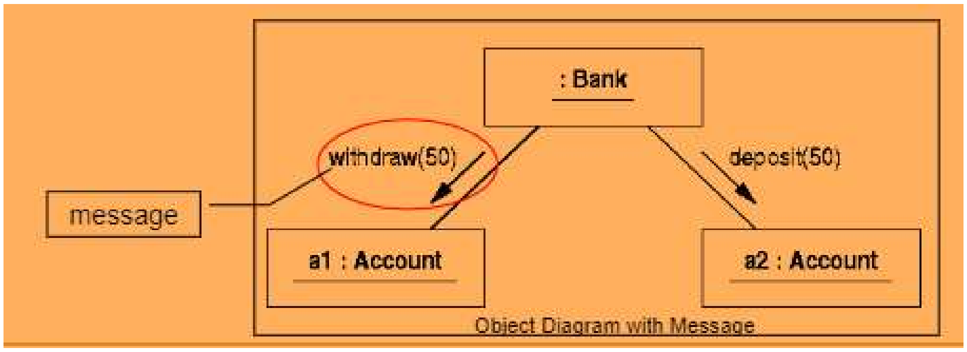
  ## Overview of This Lecture

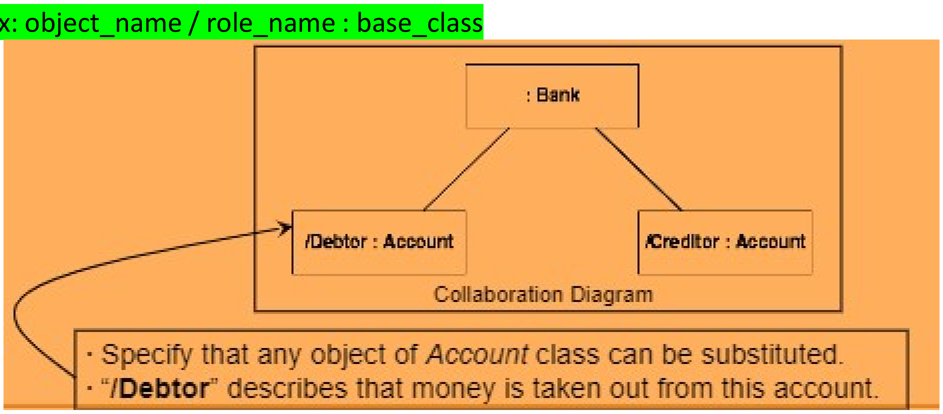
  - Interactions diagrams
    - Collaborations, classifier and association roles
    - Interaction diagrams, object creation and destruction
    - Role multiplicity and iterated messages
    - Multi-objects
    - Conditional messages, messages to self
- 

  ## Where are we now?

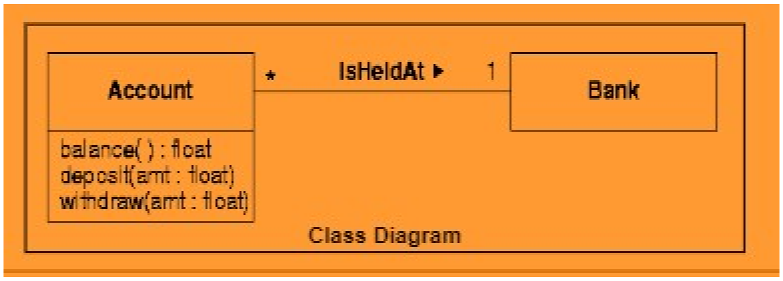
  Requirement
  Analysis
  Design
  Implement
  Test

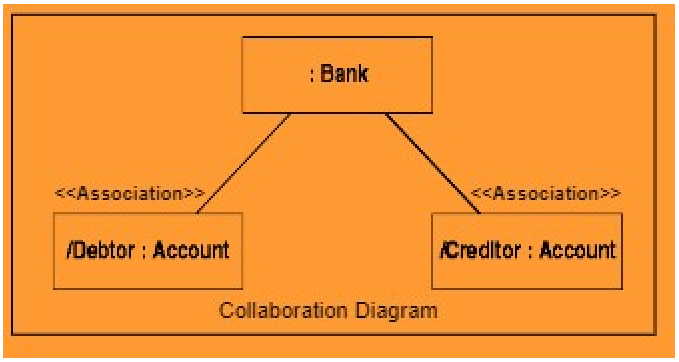
  - In depth look at the Interaction Diagrams.
- Interaction Diagrams
  - When sys running, object interact by passing messages
  - Messages define sys behavior, not shown on static diagrams (like class diagrams)
  - UML defines 2 types of diagrams for showing interactions:
    - Collaboration Diagrams
    - Sequence diagrams
- Using Object Diagram: Interaction
  - Message can be added to an Object Diagram
  - Syntax: arrow w/ message name and parameter
  - Ex:
    - The Bank performs "Funds Transfer" by *withdrawing* from one account, and *deposit* to another account
    - 

- 

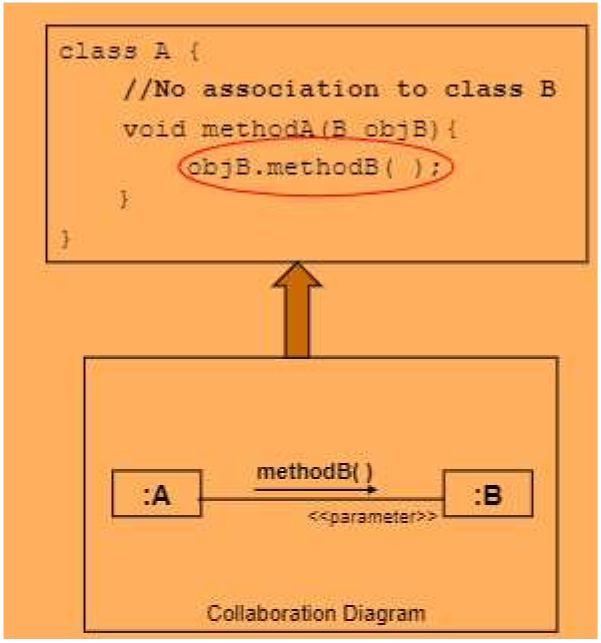
- Probs w/ Object Diagrams
  - Object diagrams show specific scenario
    - Show specific objects, not general case – can we withdraw from a2 and deposit in a1 instead?
    - Show limited number of objects and links – can we withdraw from a1 and deposit into a1 again (idk why, but sure)
    - Can't show alternative functionality – what if the withdraw causes overdraft in a1? Can we proceed w/ deposit
  - So we need smthing more general
- Solution - General method to specify behaviour
- Collaboration diagrams
  - In general , UML collaboration diagrams don't show individual object, rather the *roles* that objects can play in the interaction
  - Object diagram used to illustrate a collaboration known as *collaboration instance set*
- Classifier Roles
  - Define collaborations using classifier roles:
    - Rep any object of a class
    - Can have a name describing the role
  - Syntax: object_name / role_name : base_class
    - 

      · Specify that any object of *Account* class can be substituted.
      · "**/Debtor**" describes that money is taken out from this account.
  - Syntax Guide:
    - Classifier role not underlined, to distinguish from the object diagram usage
    - object_name can be used to label a classifier role instead of a role name, when role is not clear/important
  - Object Diagram on slide 7 is a *collaboration instance* set of the Collaboration Diagram on Slide 11:
    - Substitute an object a1 for the /Debtor role
    - Substitute an object a2 for the /Creditor role
- Roles and Objects
  - Objects can play diff roles in interactions
  - Object can be substituted for a role if
    - Its an instance of the base class of the role

        - Its one of its subclasses
    - In a given interaction, object playing certain role won't normally make use of all features given by base class of the role
        - Ex: *Account* object in the /*Debtor* role only get "withdraw()" message, but not "deposit()" message
- Association Role
    - Like classifier role, this role generalizes the links in the object diagram
    - Association role connecting 2 classifier roles indicates objects playing those roles can establish links to each other and exchange messages during interactions
- Association Stereotypes
    - 5 ways to establish link btwn 2 objects
        - Base Association
        - Parameter
        - Local Instantiation
        - Global Variable
        - Self-directed Link
    - In UML, 5 corresponding stereotypes used to denote ^those
        - <<Association>>
        - <<Parameter>>
        - <<Local>>
        - <<Global>>
        - <<Self>>
- Association Roles: Base Association
    - The most common kind of association, defined btwn corresponding classes
    - More "permanent" compared to other kind of association, usually kept as attribute in the class
    - Syntax: label the association role with stereotype <<*Association*>>
    - Ex:
    - 

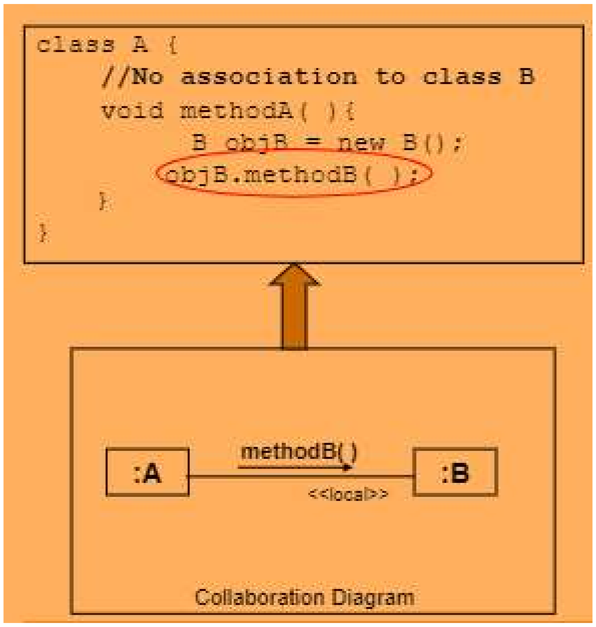
- Base Association: EX
    - 

    - Interaction possible bc the Bank object holds the **"IsHeldAt"** to the 2 account objects
    - Since most common case, can omit <association>> stereotypes
- Association Role: Parameter
    - 1 object passed to another as a parameter of a message

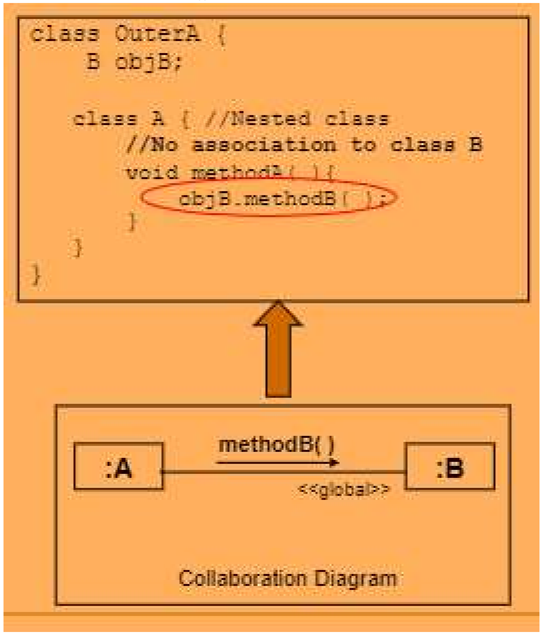
  - In pgrming languages, implemented by passing a reference to the object
  - Object getting the message knows id of parameter object, and can send messages to that object ( in the method body)
  - Link is temporary, available while operation Is executing
  - Syntax: label w/ the stereotype <<parameter>>
  - Parameter: Example
    - During the execution of methodA(), an object of class A can pass a message to an object of class B bc the reference is passed as a parameter
    - When methodA() terminates, the link will be over
    - 

```
class A {
    //No association to class B
    void methodA(B objB){
        objB.methodB( );
    }
}
```

- Association Role: Local Instantiation
  - Implementations of operations can make local instances of any class
  - Sending messages to these objects during the execution of the operation is now possible
  - Link corresponding to a local var only lasts for the duration of an operation call
  - Syntax: Label with the stereotype <<local>>
  - Local Instantiation: Ex
    - During the execution of methodA(), an object of class A can pass a message to an object of class B, bc a local object is made
    - When methodA() terminates, the link will be gone

- 

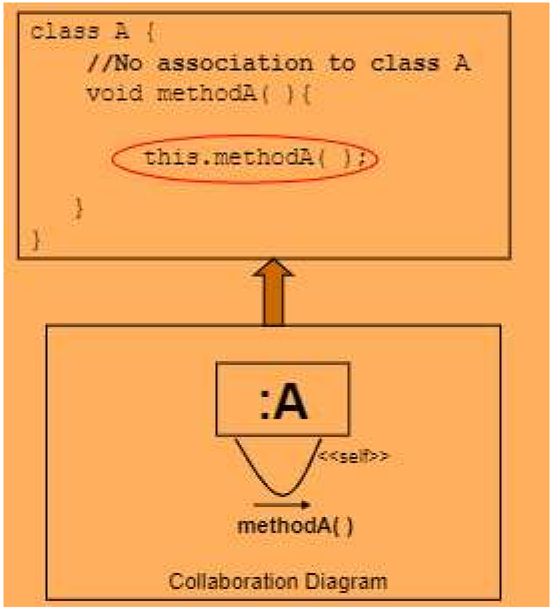
```
class A {
    //No association to class B
    void methodA( ){
        B objB = new B();
        objB.methodB( );
    }
}
```

- Association Role: Global Var
    - If any glbl vars exist and are visible, an object can send messages to an object stored in such a var
    - Ex:
        - Java -> nested class
        - C++ -> glbl object pointer
    - Syntax: Label with the stereotype <<global>>
    - Glbl Var: Ex
        - During the execution of methodA(), and object of class A can pass a message to an object of class B, bc an attribute of the parent class is accessible to all nested classes
        - When methodA() terminates, the link remains
        - 

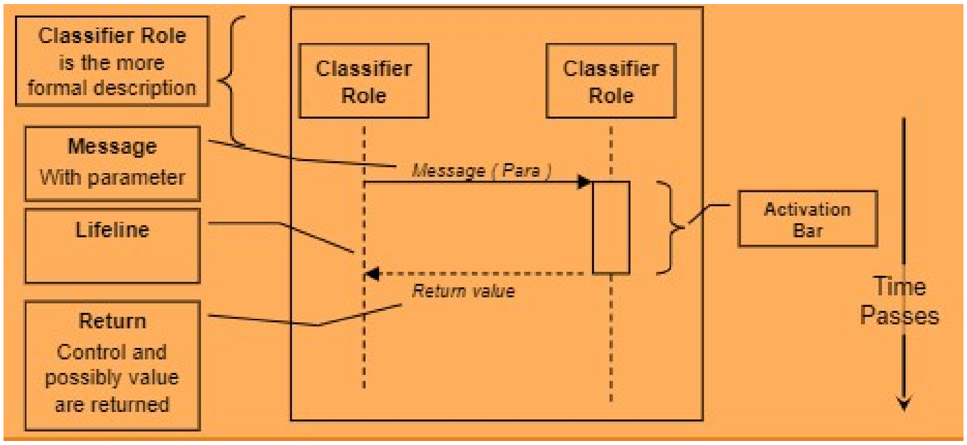
```
class OuterA {
    B objB;

    class A { //Nested class
        //No association to class B
        void methodA( ){
            objB.methodB( );
        }
    }
}
```

- Association Role: Self-Directed
    - Object can always send messages to itself, even though no explicit 'link to self' is defined
    - In pgrming langs, capability given by defining a pseudo-var called this or self
    - Syntax: Label with stereotype <<self>>
    - Self-Directed: Ex
        - During the execution of methodA(), an object of class A can send messages to itself, bc

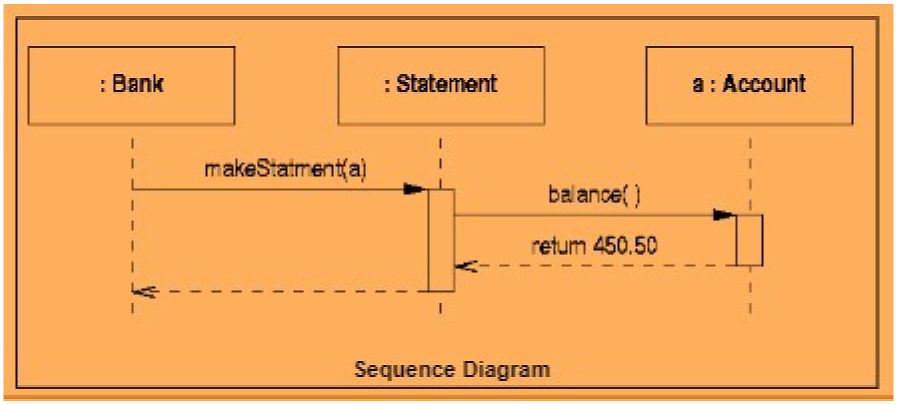
a self-reference (in java, it is keywork 'this') is always available

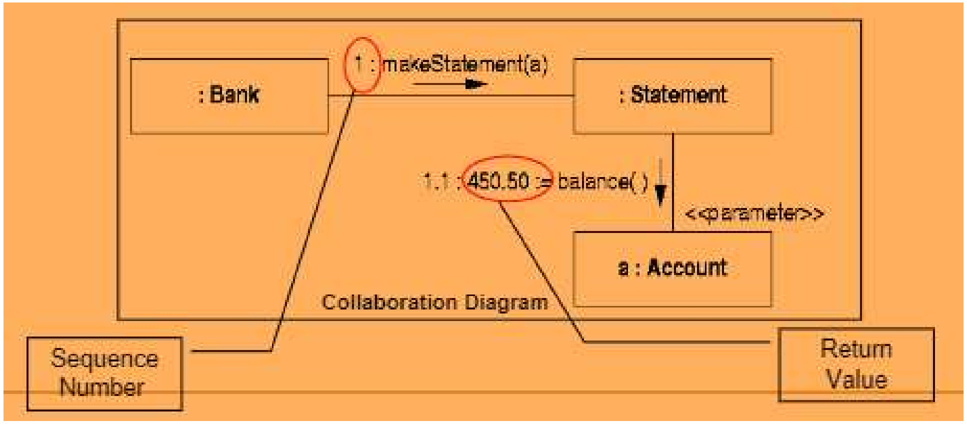
  - When methodA() terminates, the link remains
  - 

```
class A {
    //No association to class A
    void methodA( ){

        this.methodA( );

    }
}
```

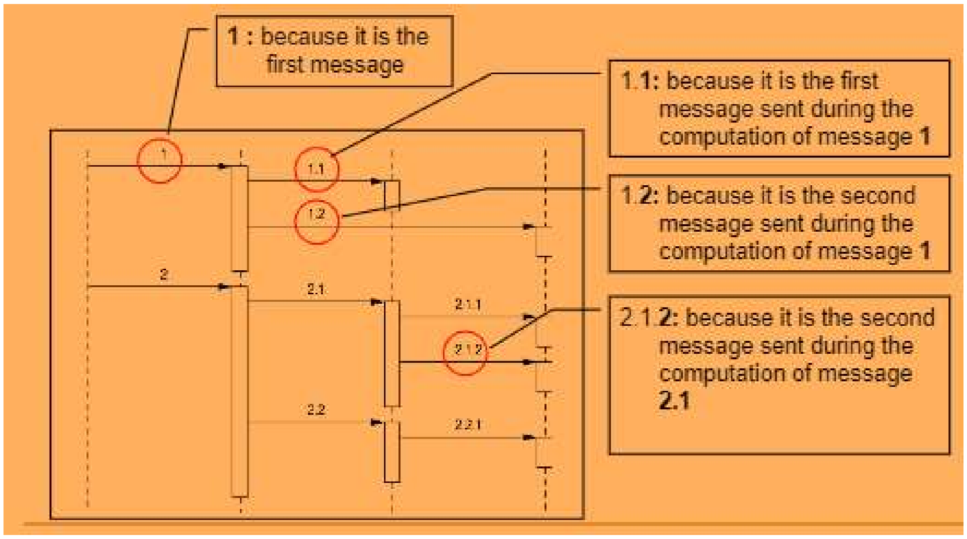
- Sequence Diagram: Review
  - Think of classifier role now
  - 

- Sequence Diagrams
  - Time when object is processing a message called activation
    - Syntax: narrow rectangle, top is connected to a message
  - When an object finishes processing a message, the ctrl returns to the sender of the message
    - Syntax: dashed arrow from the bottom of activation rectangle back to lifeline of the role that sent the message
  - The messages with solid arrowhead denote synchronous messages, like normal procedure calls (object that sends the message is suspended until the called object returns the ctrl to the caller)
  - Simple Ex:
    - *Statements* are to be printed for bank accounts: *Bank* passes the relevant *Account* to a *Statement* object for printing

        - 

        - Missing?
        - Link btwn classifier roles not indicated
            - How can a *statement* object contact the relevant *account* object
            - Have to read the diagram carefully to deduce that the link may be established by the parameter
            - Some cases, such deductions impossible/prone to error when info not enough
        - Collaboration diagram can show the same exchange, but also includes the association role used for communication
- Collaboration Diagram
    - Show classifier and association roles
    - Compared with diagram (slide 28, ^) messages also have
        - Sequence numbers to indicate order
        - Optional returned values with ':=' notation
    - 

- Collaboration vs Sequence Diagrams
    - Unlike sequence diagrams, collaborations diagrams show association role
    - Message sequence cannot be shown graphically and messages numbered to indicate the order which they sent
    - Messages can be numbered sequentially, but more commonly a hierarchical numbering scheme used (like reflect the nesting activation made explicit in sequence diagrams)
- Hierarchical Numbering
    - W/ each activation, messages numbered sequentially (start from 1)
    - Unique label can be made for each message by adding the number of the message to the end of the number of the activation sending the message
    - Syntax uses a "."
        - Used to separate the numbers
        - Used to reflect that another level of nesting of ctrl flow has been initiated
    - Hierarchical Numbering Ex:

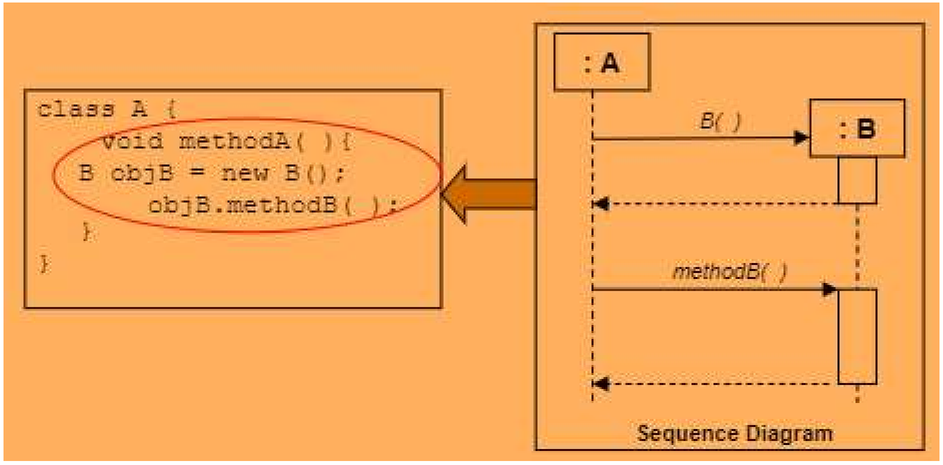
- ▪ 

1 : because it is the first message

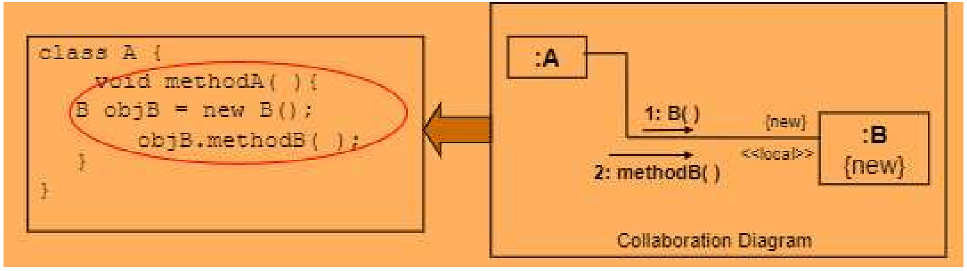
1.1: because it is the first message sent during the computation of message 1

1.2: because it is the second message sent during the computation of message 1

2.1.2: because it is the second message sent during the computation of message 2.1

- Interaction Diagram: Additional Notations
  - ○ Subsequent ex illustrate the notations for:
    - ▪ object creation
    - ▪ Object destruction
    - ▪ Iterated messages
    - ▪ Multiobjects
    - ▪ Conditional messages
    - ▪ Message to self
  - ○ Need to take note of how to protrait certain interactions in both SD (sequence diagrams) and CD (collab diagram)
- Sequence Diagram: Object Creation
  - ○ Time is explicitly reped in a sequence diagram, the object creation is easy to draw
  - ○ 

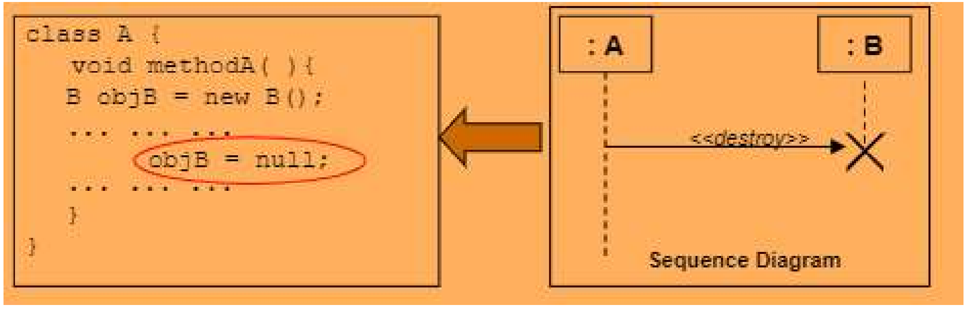
```
class A {
   void methodA( ){
  B objB = new B();
       objB.methodB( );
   }
}
```

- Collaboration Diagram: Object creations
  - ○ In collaboration diagram, new objects and new associations have to be labeled with {new}
  - ○ 

```
class A {
    void methodA( ){
  B objB = new B();
        objB.methodB( );
    }
}
```

- Sequence diagram: Object Destruction
  - ○ In langs w/ auto garbade collection (java), can't explicitly delete an object
  - ○ Instead, remove all references to the object for auto garbage collection

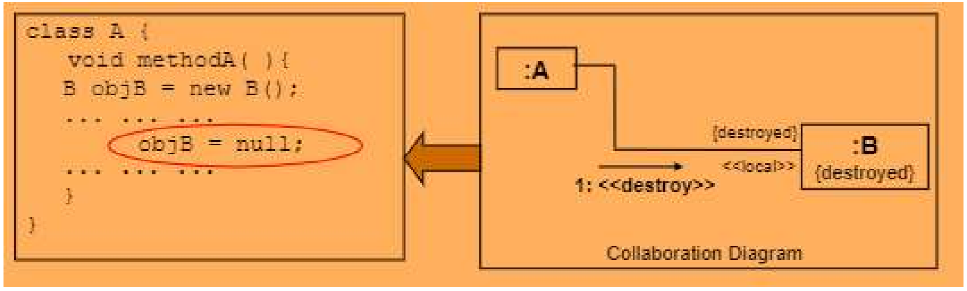
- ○ Label the message <<destroy>>
- ○



```
class A {
   void methodA( ){
   B objB = new B();
   ... ... ...
       objB = null;
   ... ... ...
   }
}
```

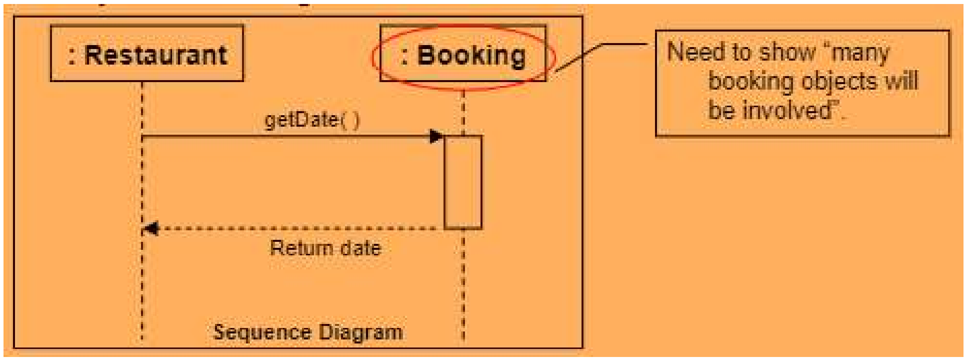
- ○

- Collaboration Diagram: Object Destruction
  - ○ In a Collaboration Diagram:
    - Similar, label messages <<destroy>>
    - Label the destroyed objects and links with *{destroyed}*
    -

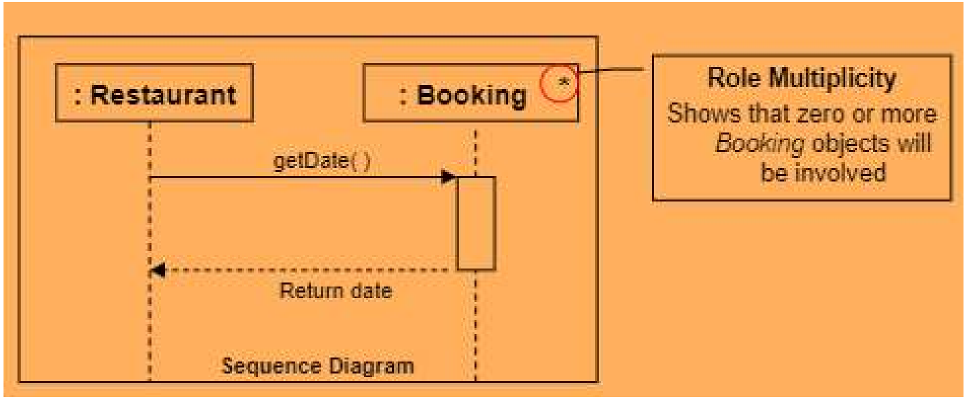


```
class A {
   void methodA( ){
   B objB = new B();
   ... ... ...
       objB = null;
   ... ... ...
   }
}
```

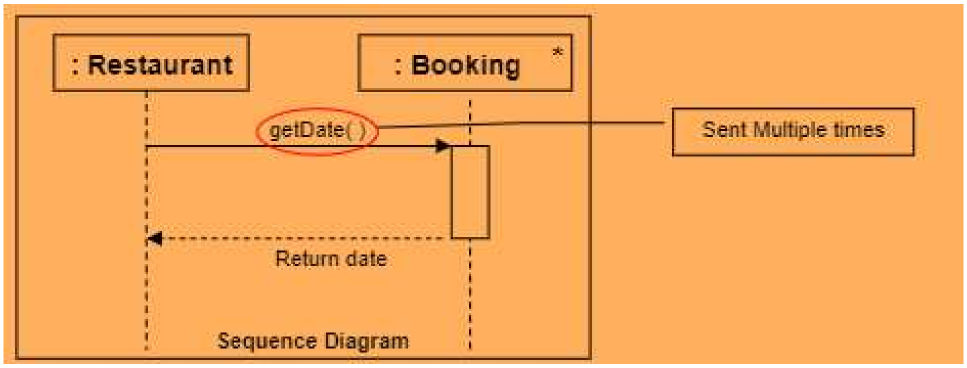
- Role Multiplicity
  - ○ Numb of objects playing a role can vary from 1 occasion to another
  - ○ Ex: restaurant case study
    - Looking for bookings for a certain date depends on how many  total bookings are there
    -

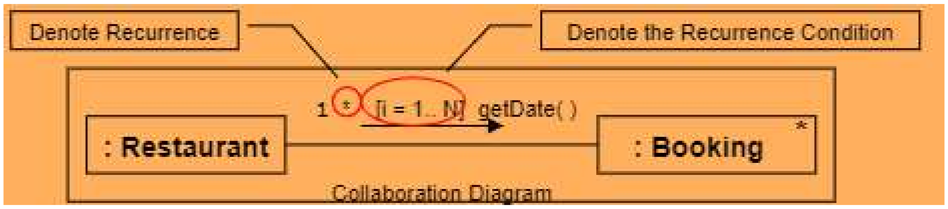


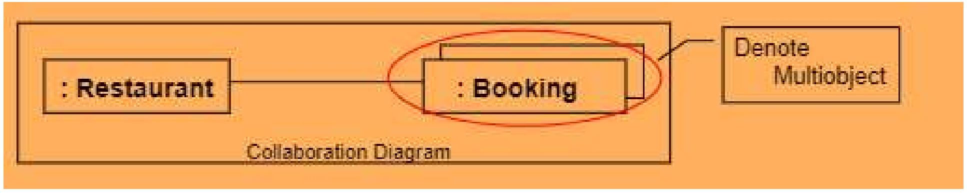
  - ○ Roles mult can be added to a classifier role to indicate numb of objects involved
  - ○ Syntax: same as the multiplicity notation in class diagram (e.g., 1...8,*,2..*,etc)
  - ○

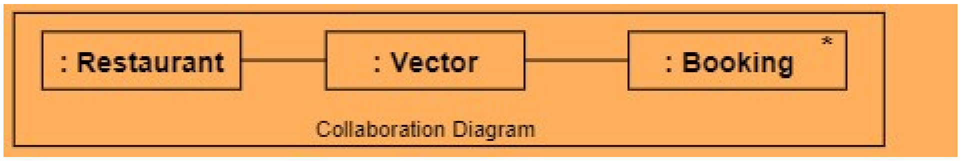


  - ○ Notation is same as both *Sequence Diagram* and *Collaboration Diagram*
  - ○ However, the fact that the message is iterated is still not represented

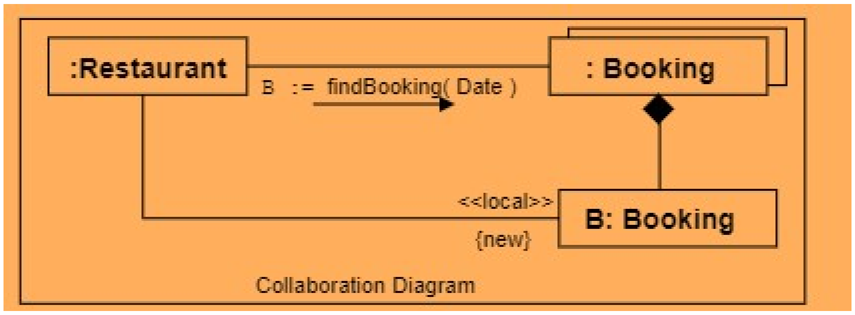
- 

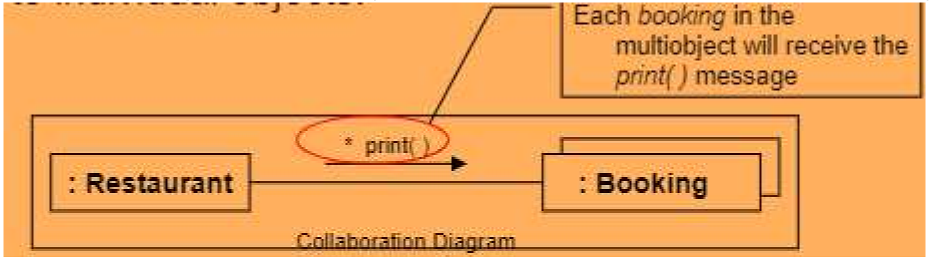
- Iterated Messages
    - Clarify this by:
        - Adding a multiplicity to the affected role
        - Specifying that the message is iterated
    - Syntax: recurrence consists in '*' written after the sequence number, possibly followed by an iteration clause, no formal syntax for iteration clause, Pseudo code-like condition usually used, (e.g., [i = 1...N] or [i = 1 to N]
    - Ex:
        - 

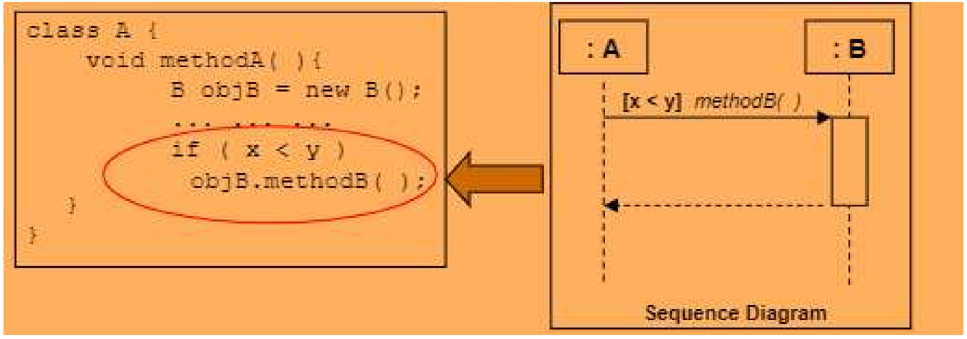
- Multiobject
    - Multiobject denotes collection of objects: it is a role w/ a multiplicity of 0 or more
        - 

    - Implies an intermediate data structure
        - 

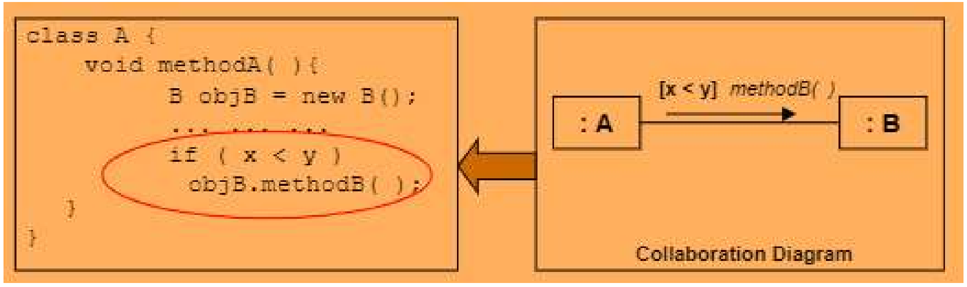
- Property of Multiobject
    - Using a multiobject prevents a premature commitment to a particular data structure:
        - Ex: What if *vector* is not a good data structure for this case?
    - Semantically, a mutliobject is a **single** object rep a collection of objects
    - Single message sent to it implies an operation involving the collection of objects
    - Good ex of this op is looking for a certain object in collection
- Multiobject Ex:
    - Assume there is only one booking per date to simplify the discussion
        - Singl message *findBooking(Date)* sent to the Booking multiobject
        - Multiobject inspects all its *Booking* objects and returns the appropriate booking *B* (to indicate that B is not a new object, but one from the multiobject, composition link used)
        - *Restaurant* stores B as a local reference for subsequent ops

- 

- Multiobjects: Message for all Objects
  - Send a message to all objects in the multiobject:
    - Send single message to the multiobject
    - Multiobject goes thru some iterative process and sends the message to each object in the collection
  - By conversion, such interactions can be abbreviated by using iterative messages
  - Iterated messages to a multiobject are understood to be sent to individual objects
  - 
Each *booking* in the multiobject will receive the *print( )* message
* print( )
: Restaurant
: Booking
Collaboration Diagram

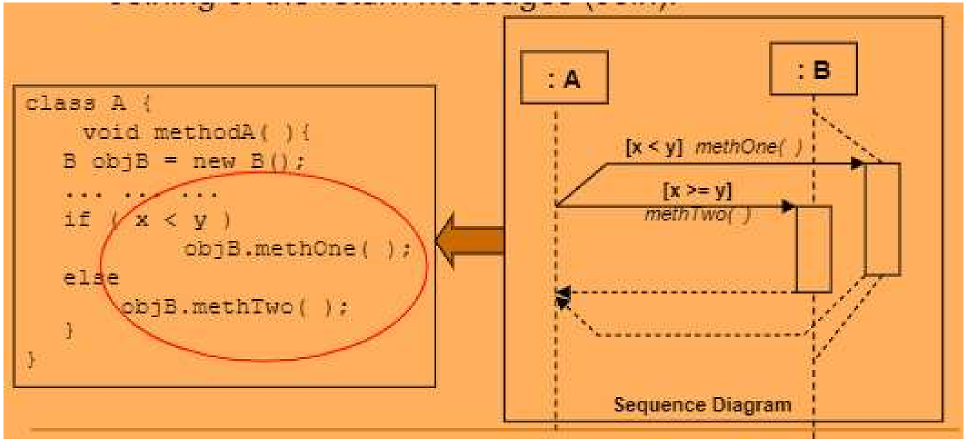
- Sequence Diagram: Conditional Message
  - *Conditions* can be added to messages to show the situations when they are sent
  - Syntax: Write the condition in [] preveding a message
  - 

```
class A {
    void methodA( ){
        B objB = new B();
        ... ... ...
        if ( x < y )
          objB.methodB( );
    }
}
```

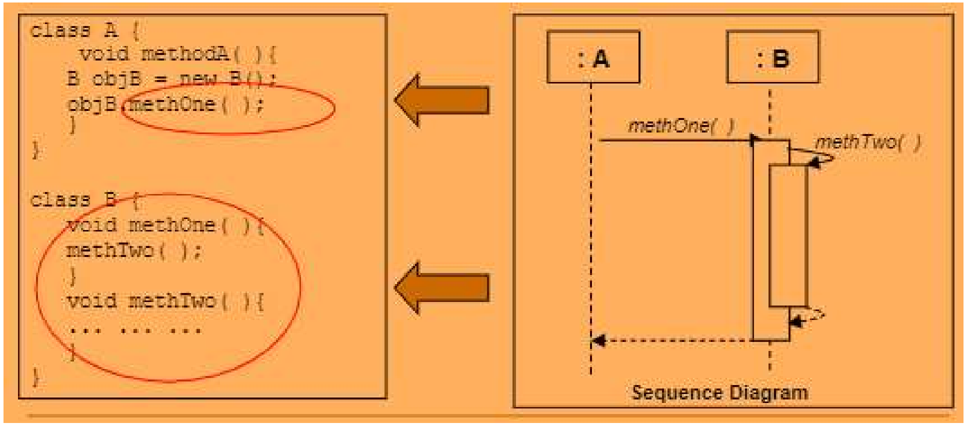
- Collaboration Diagram: Conditional Message
  - Same syntax can be used for a collaboration diagram
  - 

```
class A {
    void methodA( ){
        B objB = new B();
        ... ... ...
        if ( x < y )
          objB.methodB( );
    }
}
```

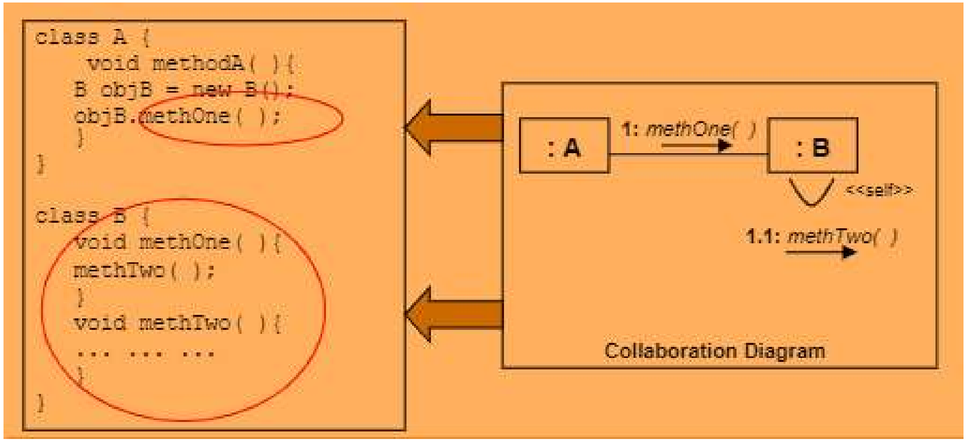
- Alternative Flows
  - Sequence diagrams can show alternative message sequences in one diagram:
    - 2 or more messages start at same point (fork)
    - They are distinguished by conditions (only one will be sent)
    - Return messages come together later (join)
    - Objects that receive messages may need branching lifelines to rep alternative possibilities

   - Should draw 2 sep diagrams instead
- Sequence Diagram: Alternative flow
   - Pay attention
   - Branching of the messages = fork
   - Joining of the return messages = join
   - 

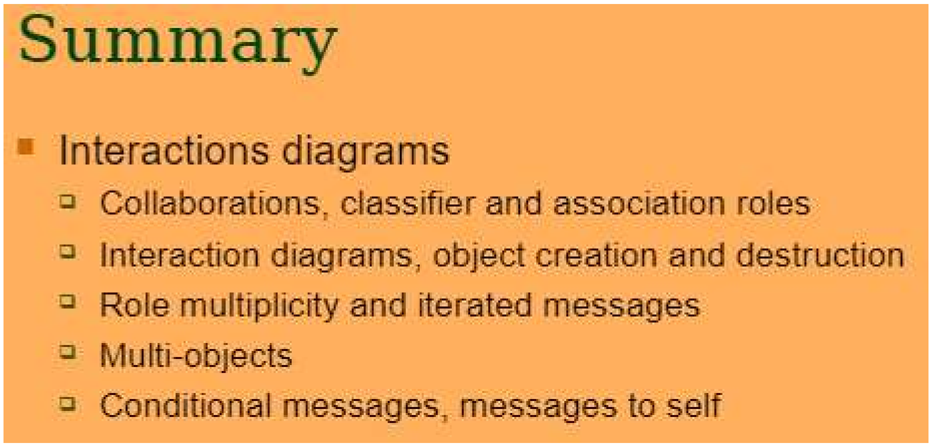
```
class A {
    void methodA( ){
  B objB = new B();
  ... ... ...
  if ( x < y )
          objB.methOne( );
  else
      objB.methTwo( );
  }
}
```

- Sequence Diagram: Message to Self
   - An object can send message to itself: invoking another operation on its own
   - Usually reps implementation details
   - 

```
class A {
    void methodA( ){
  B objB = new B();
  objB.methOne( );
  }
}

class B {
  void methOne( ){
  methTwo( );
  }
  void methTwo( ){
  ... ... ...
  }
}
```

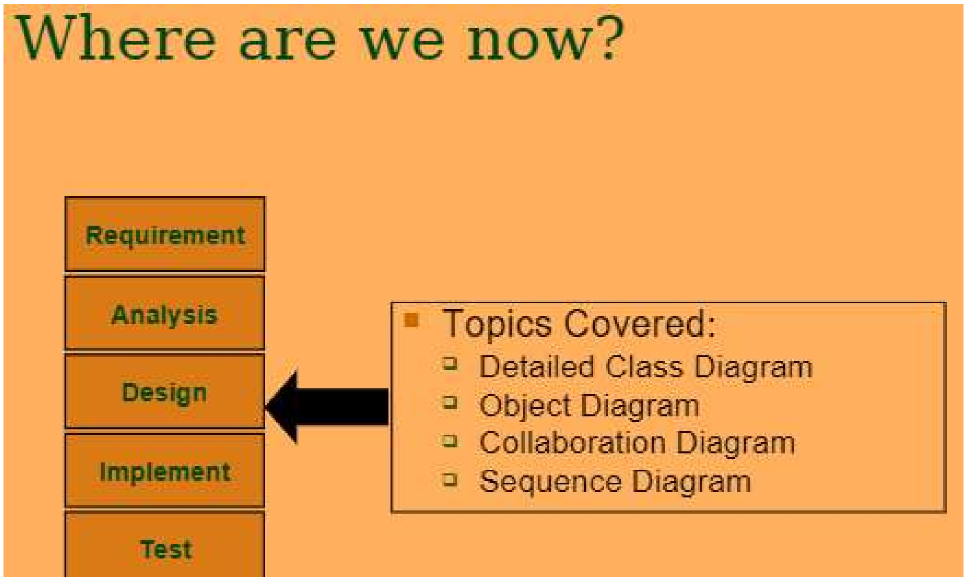
- Collaboration Diagram: Message to Self
   - Make use of the <<self>> stereotype for collaboration diagram
   - 

```
class A {
    void methodA( ){
  B objB = new B();
  objB.methOne( );
  }
}

class B {
  void methOne( ){
  methTwo( );
  }
  void methTwo( ){
  ... ... ...
  }
}
```



# Summary

- Interactions diagrams
  - Collaborations, classifier and association roles
  - Interaction diagrams, object creation and destruction
  - Role multiplicity and iterated messages
  - Multi-objects
  - Conditional messages, messages to self



# Where are we now?

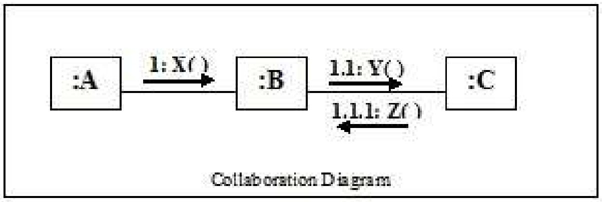
- Topics Covered:
  - Detailed Class Diagram
  - Object Diagram
  - Collaboration Diagram
  - Sequence Diagram

# L7 Assessment

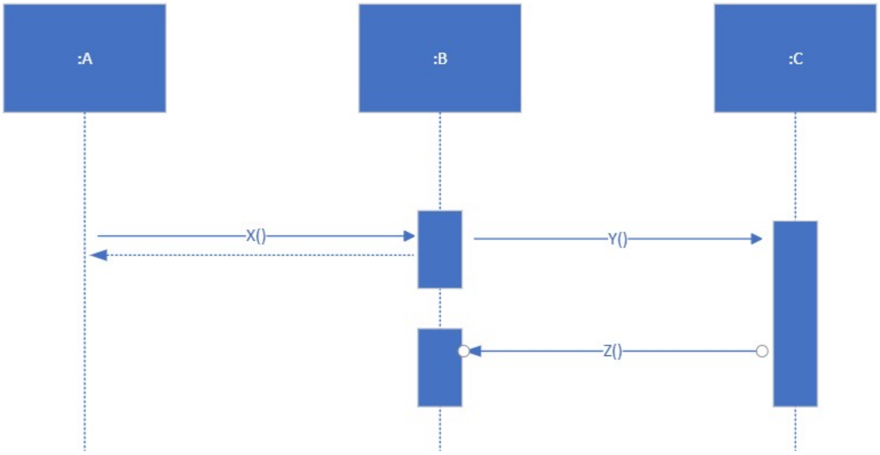
Thursday, March 2, 2023 9:35 AM

**Question 1**
**10 Points**

Draw a sequence diagram corresponding to the collaboration diagram below:

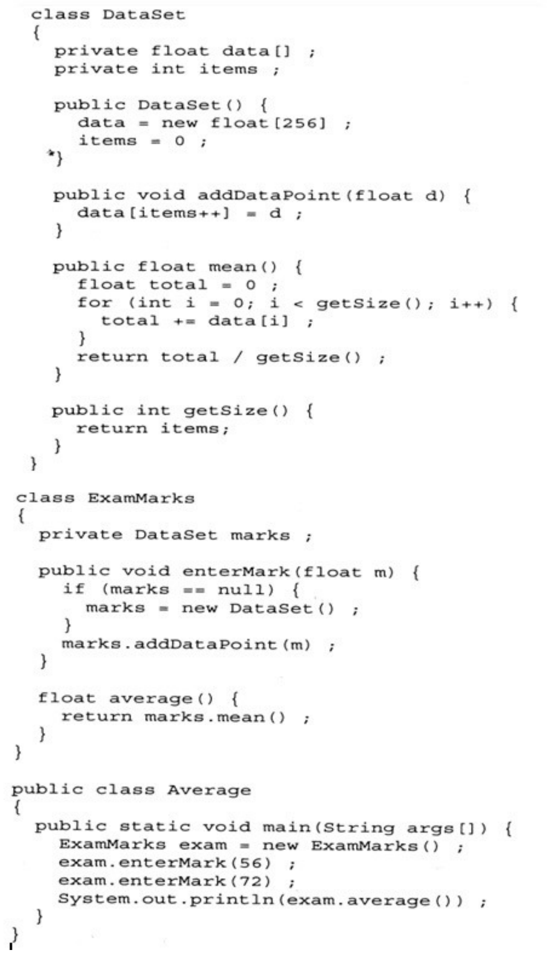


Collaboration Diagram



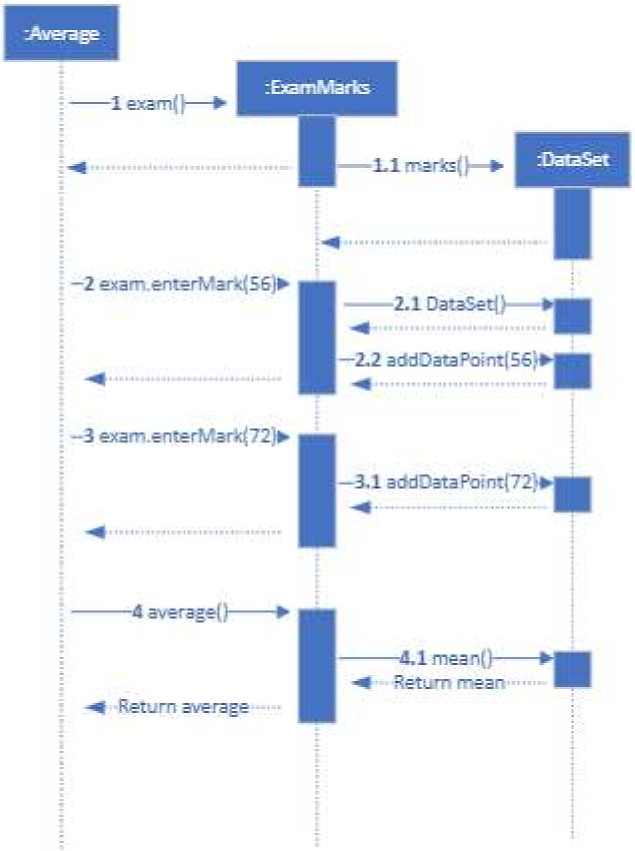
**Question 2**
**10 Points**

**(Exercise 9.7 of [Priestley; 2004])** The code below shows a class DataSet, which provides basic statistical functionality on a set of data, and a class ExamMarks, which uses DataSet to store and work out the average of a set of exam marks. The main function shown reads in two marks and uses ExamMarks to store them and print out the average. Draw a sequence diagram showing the interaction that takes place when the main function executes.



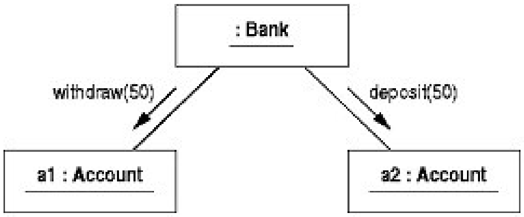
```
class DataSet
{
  private float data[] ;
  private int items ;

  public DataSet() {
    data = new float[256] ;
    items = 0 ;
 *}

  public void addDataPoint(float d) {
    data[items++] = d ;
  }

  public float mean() {
    float total = 0 ;
    for (int i = 0; i < getSize(); i++) {
      total += data[i] ;
    }
    return total / getSize() ;
  }

  public int getSize() {
    return items;
  }
}

class ExamMarks
{
  private DataSet marks ;

  public void enterMark(float m) {
    if (marks == null) {
      marks = new DataSet() ;
    }
    marks.addDataPoint(m) ;
  }

  float average() {
    return marks.mean() ;
  }
}

public class Average
{
  public static void main(String args[]) {
    ExamMarks exam = new ExamMarks() ;
    exam.enterMark(56) ;
    exam.enterMark(72) ;
    System.out.println(exam.average()) ;
  }
}
```

A:



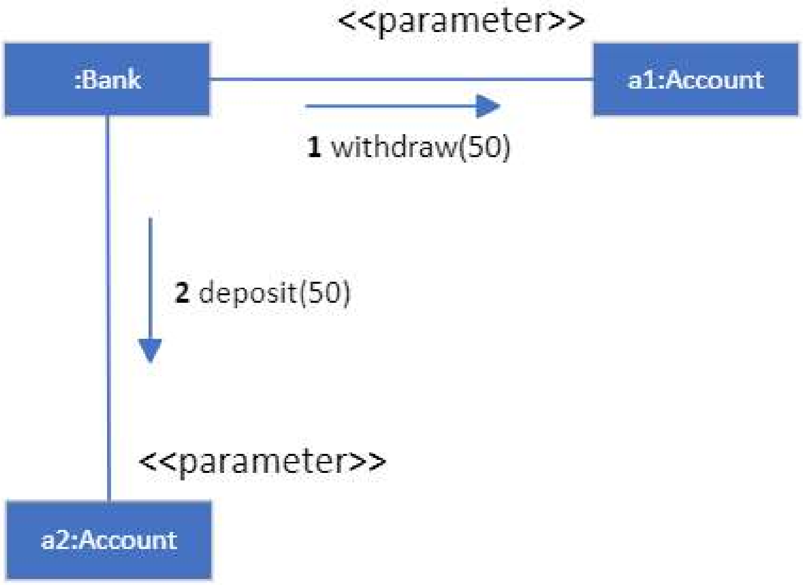
**Question 3**
**10 Points**

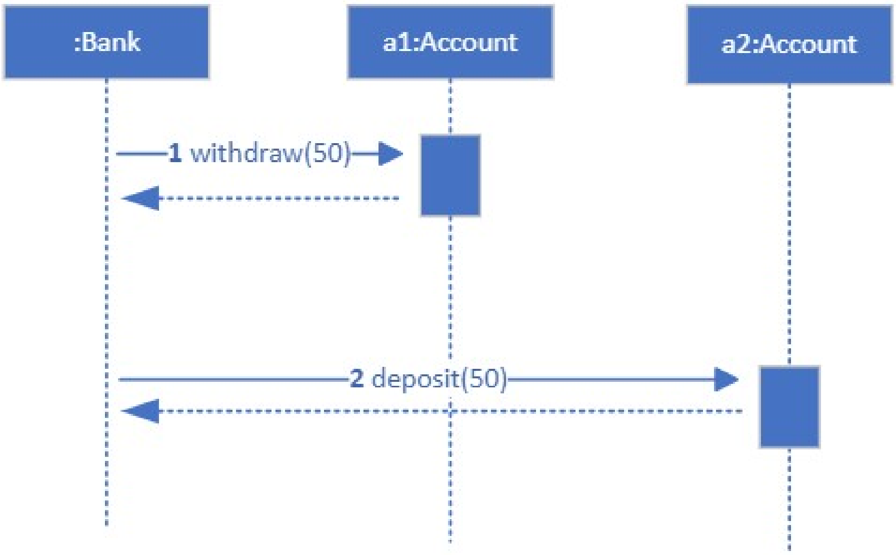
**(Exercise 9.1 of [Priestley; 2004])** Suppose that in a banking system a transfer is carried out in the following way: a transfer object is created to control the interaction, and the two accounts and the amount to be transferred are then passed as parameters to a 'doTransfer()' method in the transfer object. Draw a collaboration diagram, based on the below object diagram, illustrating this interaction. Draw a sequence diagram showing the same interaction and discuss which is the most suitable diagram in this case.



A: The collaboration diagram is more suitable because it is clearer, and it shows the steps in a similar way to the object diagram and the sequence diagram without showing more than needed.

A:

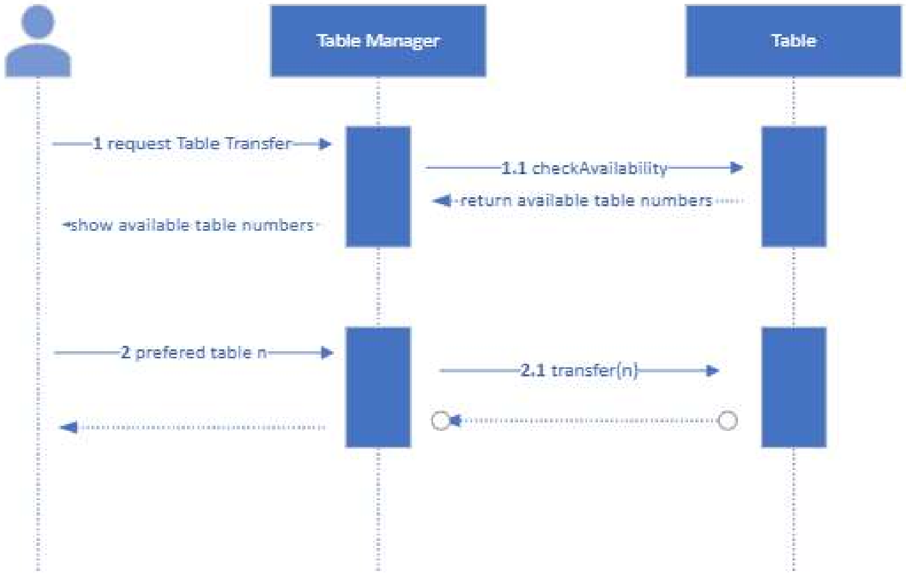




**Question 4**
**10 Points**

(Exercise 5.8 of [Priestley, 2004]) Produce a sequence diagram showing a realization of the basic course of events for the Table Transfer use case (that is, transferring a customer from one table to another). Assume that a table number is provided as a parameter for a system message transfer( ) and show on your diagram how the table corresponding to this number is identified.

A:



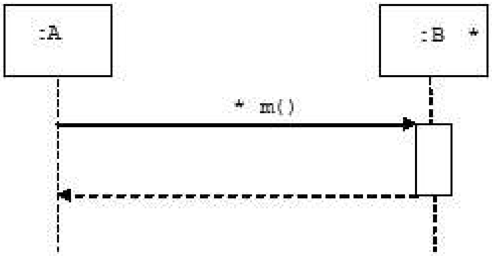
**Question 5**
**10 Points**

What is true regarding the classifier roles?

**Two classifier roles cannot have the same base class;**
**A classifier role is a class;**
**A classifier role is an object;**
**A classifier role defines the role that objects can play in interactions;**
**None of the above.**

**Question 6**
**10 Points**

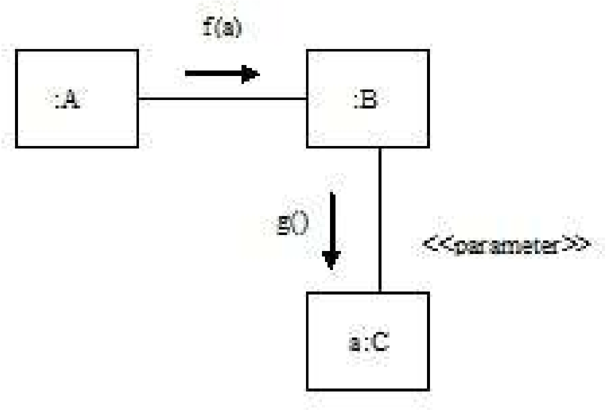
Given the below sequence diagram, what is the meaning of two '*'s?



**They mean that method m() contains a Vector/Array data structure;**
**They mean there are zero or more associations roles involved;**
**The '*' is the box means there are zero or more objects of class B involved, and the other one means method m() is called many times;**
**The above sequence diagram is wrong as '*' appears twice in different contexts;**
**None of the above.**

**Question 7**
**10 Points**

Given the below UML diagram, which of the following statements are true?



**The stereotype <<parameter>> is incorrectly defined in the object diagram;**
**The message passing 'f(a)' is incorrectly defined in the class diagram;**
**The generalization relationship between B and C is incorrectly specified;**
**Object 'a' is passed as a parameter of a message;**
**None of the above.**