Lecture 06

Interaction Diagrams

Session Outcomes

- Describe dynamic behavior and show how to capture it in a model.
- Illustrates how objects interact with each other.
- Emphasizes time ordering of messages.
- Model simple sequential flow, branching, iteration and concurrency.

The Story So Far…

- Talk to the clients to gather **requirements**.
- Draw a **use case diagram** as an overview.
- Write the main scenario and extensions for each use case:
  - interaction between system and actors.
- Generate **CRC cards** from the use cases.
- Generate a **class diagram** from the CRC cards:
  - a static view of the system at compile-time.

What Happens Next?

- Generate **interaction diagrams** from the class diagram and use case scenarios:
  - **a dynamic view** of the system at **run-time**;
  - for each scenario there is a diagram showing instances and interactions that realize the scenario at run-time.
- **A use case scenario** tells **what** happens:
  - it is a “black box” description of the system.
- **An interaction diagram** tells **how** it happens:
  - it is a “white box” description of the system.
  - however, it is NOT an algorithm!

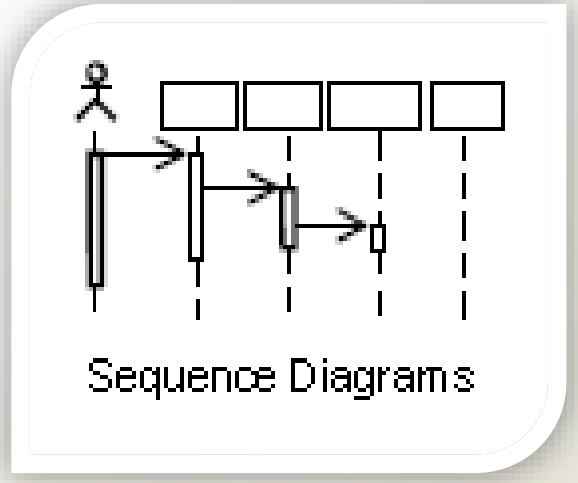
Interaction Diagrams

- It describe how groups of ***objects*** interact in some behavior.
- Typically, captures the behavior of a single use case.
- Two kinds of interaction diagrams:
  - Sequence diagrams
  - Communication diagrams-(Collaboration diagram)

Sequence Diagram

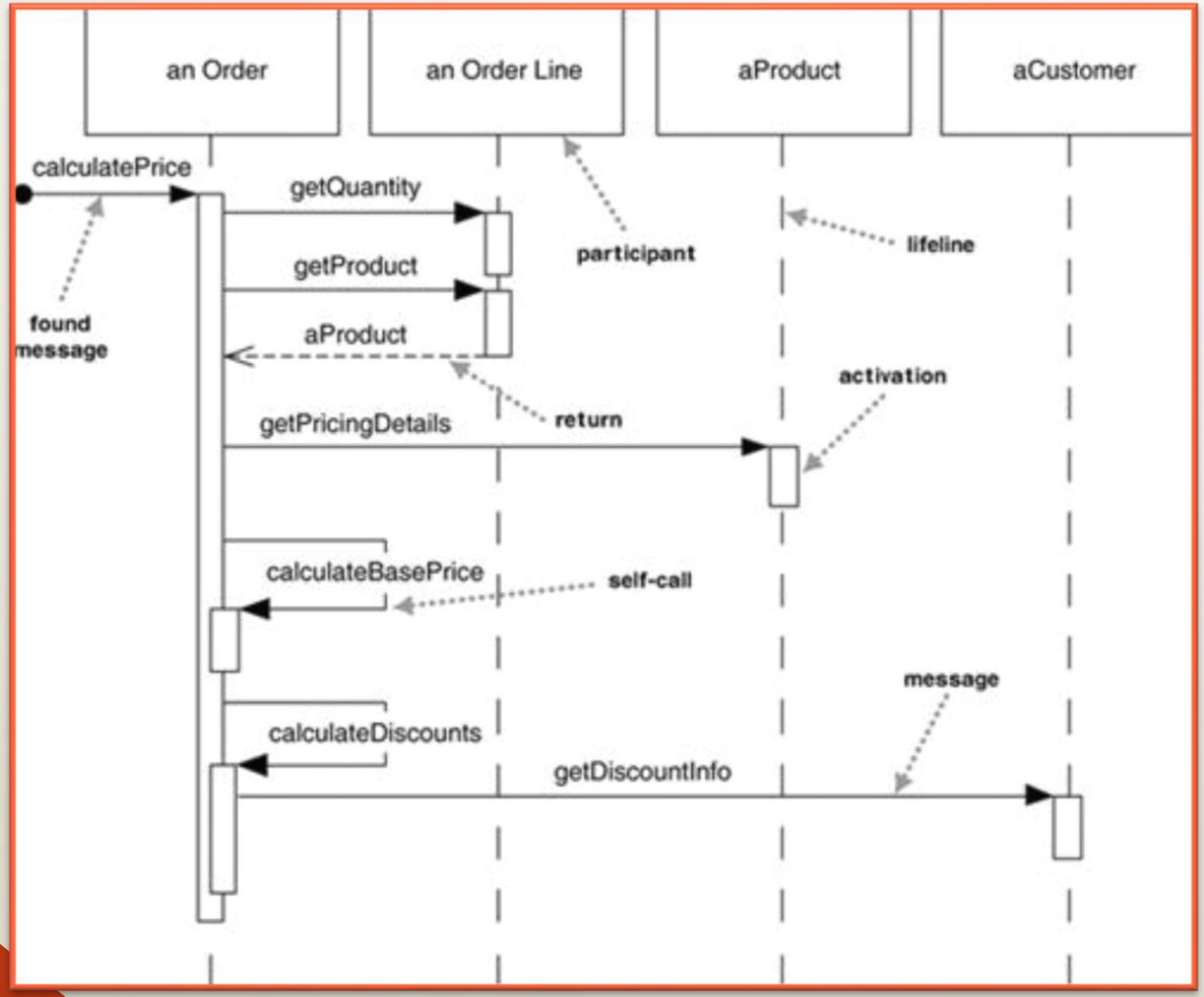
What Is a Sequence Diagram?

- A sequence diagram is an interaction diagram that emphasizes the time ordering of messages.
- Sequence Diagrams are more often used than Communication Diagrams.



- The diagram shows:
  - The objects participating in the
  - interaction.
  - The sequence of messages exchanged.

Sequence Diagram - Example



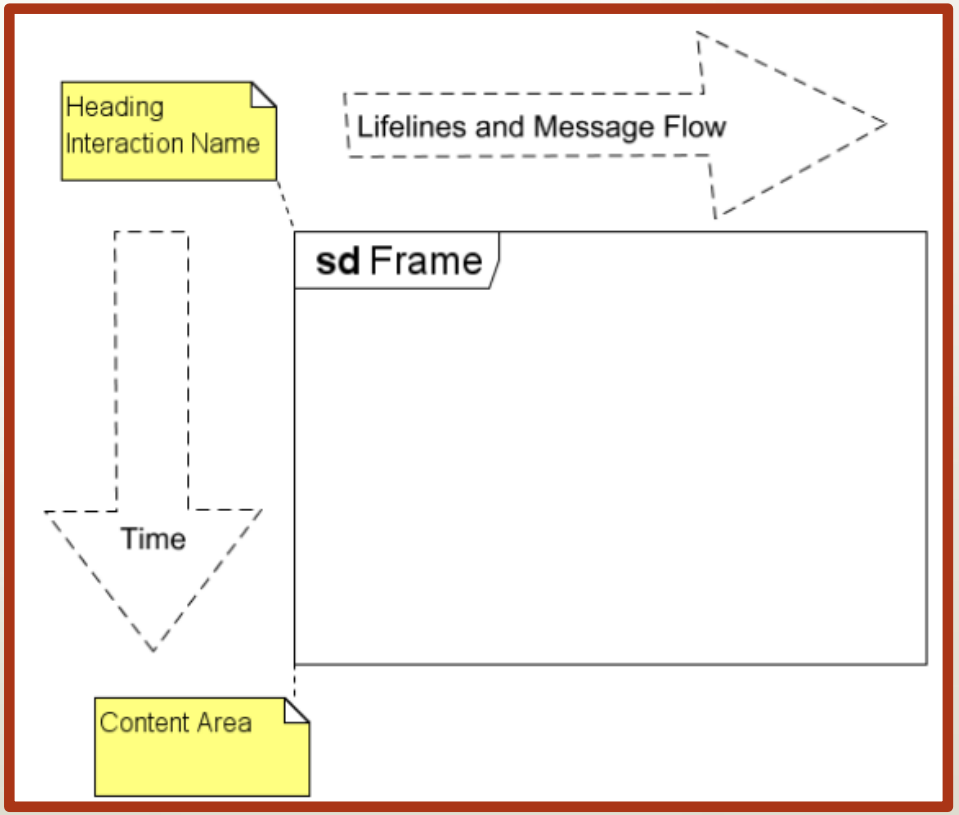
Elements of a Sequence Diagram

- Frame
- Actor
- Objects
- Life line
- Messages
- Execution

Frame

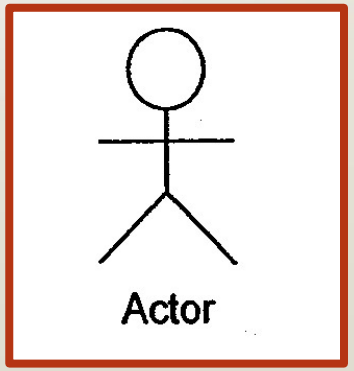
- A frame represents an interaction, which is a unit of behavior that focuses on the observable exchange of information between different objects.
- **sd** → Sequence Diagram.

<sd name of the diagram>



Actor

- A person or system that derives benefit from and is external to the system.
- Participates in a sequence by sending and/or receiving messages.
- Actor in a Sequence diagram
- Still represented by the stick figure as on use case diagram
- **don't have to place in any particular order across the top of the diagram** but **better to organize** them in the order in which they participate in the sequence



Objects

- Objects are shown in the same way as in UML object diagrams.
- A named object

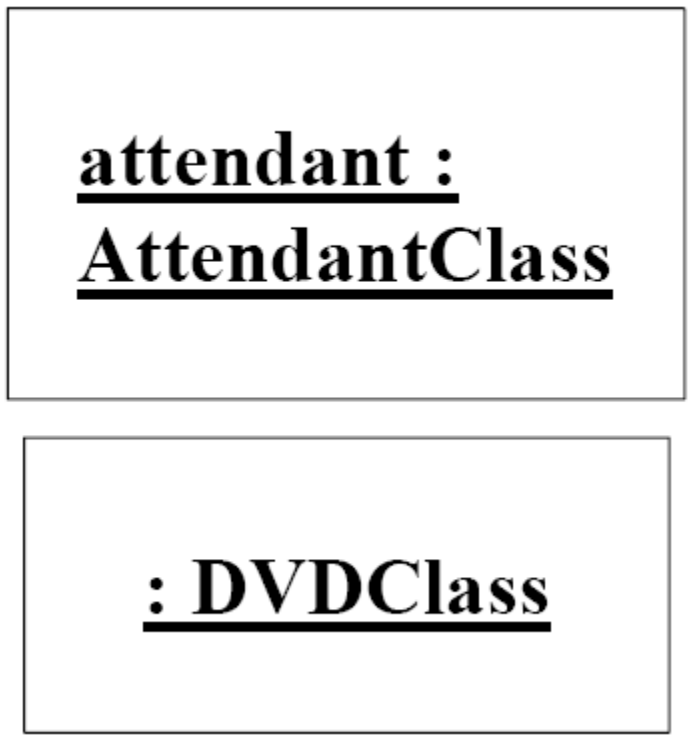
  (Named instance of a class)

Object name : class name

- An anonymous object

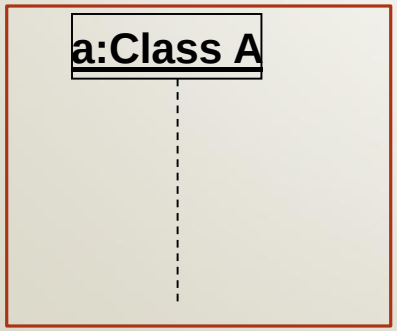
(Anonymous instance of a class)

: class name

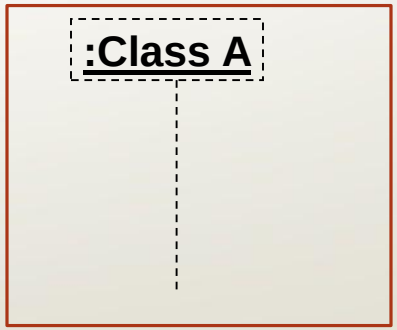


Life Line

- The *Life-Line* represents the object's life during the interaction.
- A Lifeline is shown using a symbol that consists of a rectangle forming its "head" followed by a vertical line (which may be dashed) that represents the lifetime of the participant.
- **Anonymous lifeline** has no name - arbitrary representative of class.



***Lifeline** with name "a" of class A*



***Anonymous lifeline** of class A.*

Elements of Sequence Diagram

- Objects
- Life line
- Messages
  - **Synchronous call**
  - **Asynchronous call**
  - **Create**
  - **Delete**
  - **Reply**

Messages

- **Message** is a **named element** that defines a specific kind of communication between **lifelines** of an interaction.
- Can include parameters in the messages on a sequence diagram.
- A message is shown as a line from the sender message end to the receiver message end.
- The send and receive events may both be on the same lifeline.

Message Passing

- Message passing is the **invocation of a method** in one object, by another method that belongs to a different object.
- When a message is sent from object A to object B:
  - object A asks object B to execute one of its methods (i.e. invoke the method);
  - object B receives the message and executes the method;

Message Labels

- A message between two objects is labeled with:
  - The **name** of the method belonging to the receiving object;
  - Optionally, any **arguments** passed from the sending object to the receiving object.
- Example : setTitle() is a method that belongs to video (the receiving object) and is invoked by inventory (the sending object).

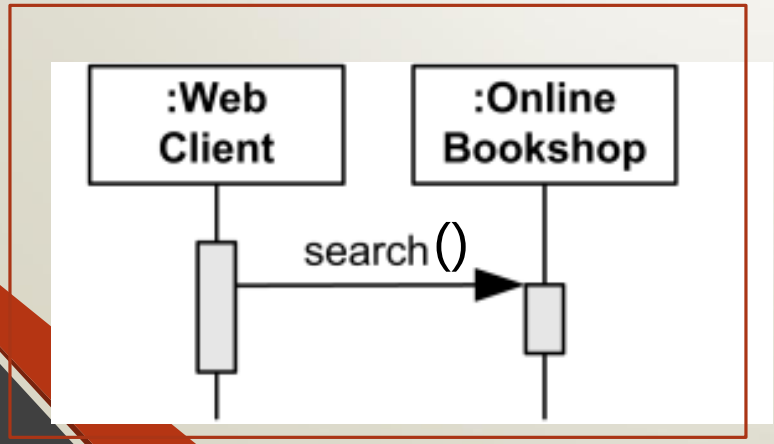
Messages by Action Type

- Synchronous call
- Asynchronous call
- Create
- Delete
- Reply

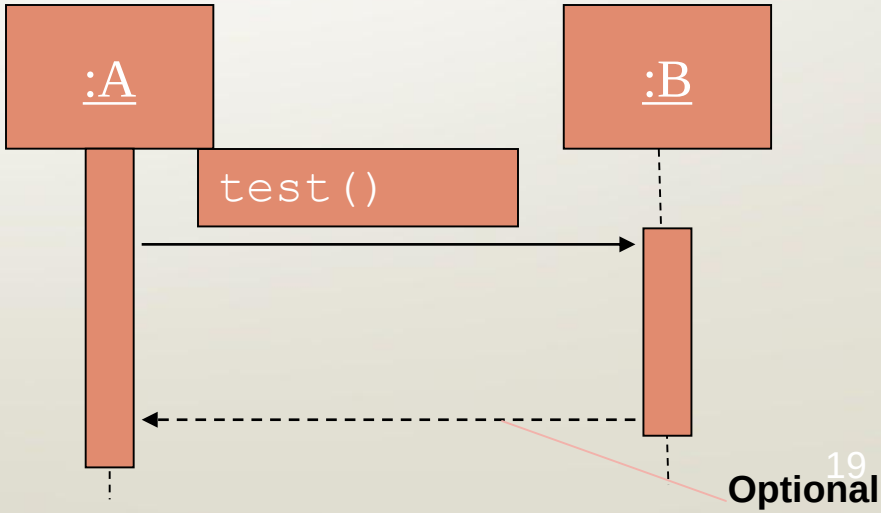
Synchronous Call

- Synchronous message is used when the sender waits until the receiver has finished processing the message.(Also known as call - send message and it will suspend execution while waiting for response)

Web Client searches Online Bookshop and waits for results



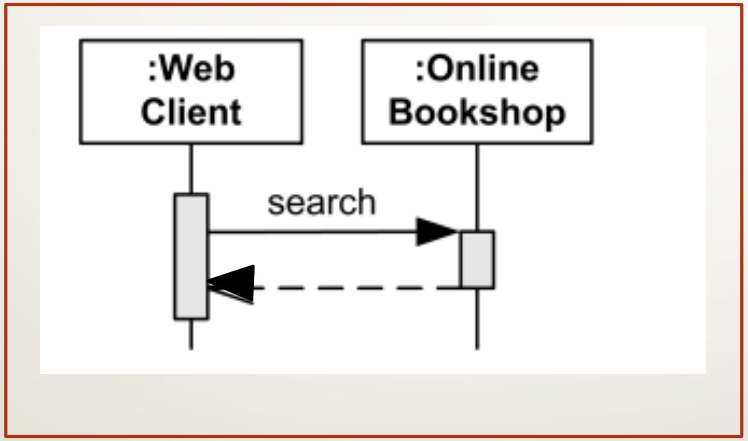
Preferred if the return value is of importance



Reply Message

Reply message to an operation call is shown as a dashed line with arrow head.

In UML 2.4,return message of the synchronous call is also known as Reply message.



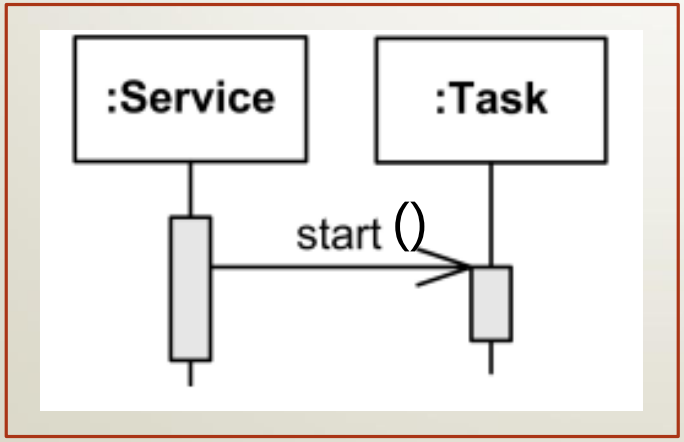
Web Client searches Online Bookshop and waits for results to be returned

Asynchronous Call →

- **Asynchronous call -** The sender does not wait for the receiver to finish processing the message (for any return values), it continues immediately.

- An open arrowhead is used to asynchronous call.

Service starts Task and proceeds in parallel without waiting

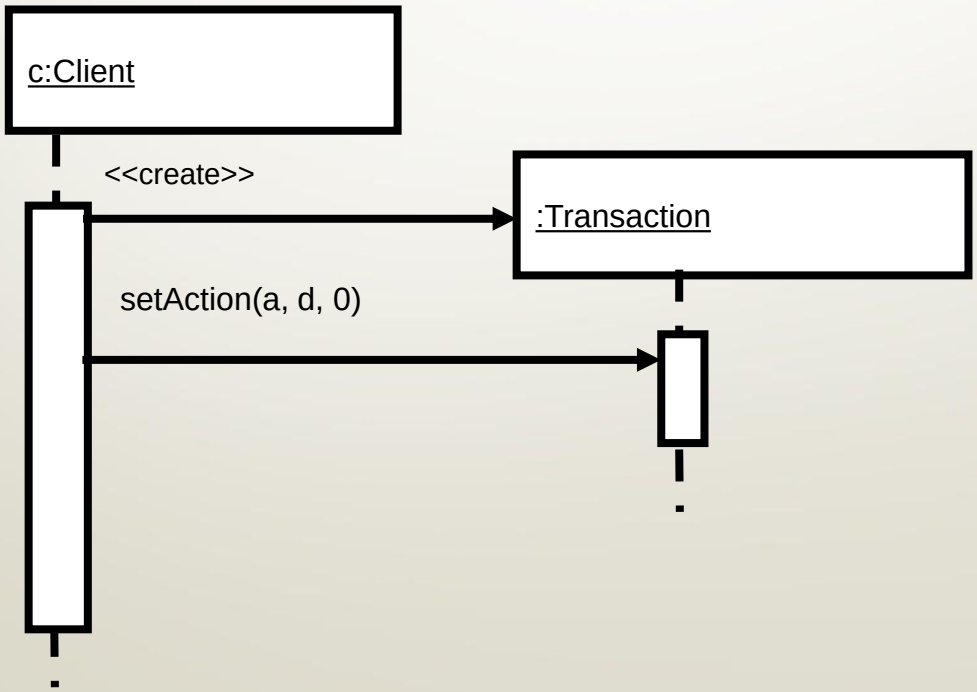


Create and Delete Messages - (Create and Destroy)

- In Java and **C++ constructors** are necessary to create new objects.
- **In C++ destructors** are necessary to destroy objects:
  - programmers must explicitly remove used objects from memory, otherwise leaks occur;
  - C++ provides a default destructor for each class, which can be overridden to incorporate extra tasks.
- However, **Java** provides **automatic garbage collection** to clean up the mess:
  - programmers do not have to destruct used objects.

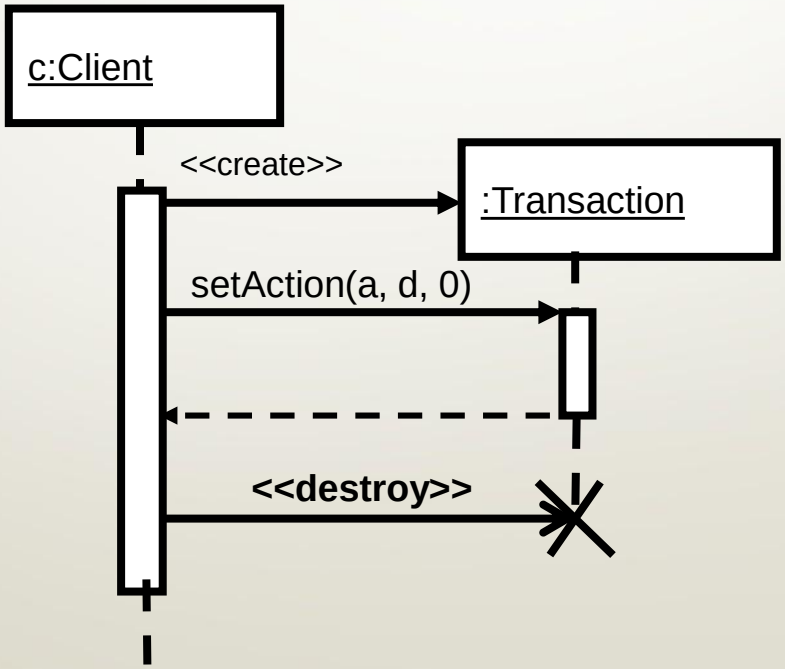
Create message

- Create message is sent to lifeline head to create itself.
- Create message is shown as colored arrowhead with <<create>> keyword pointing to created lifeline's head.



Delete Message

- **Delete** message is sent to terminate another **lifeline**.
- The lifeline usually ends with a cross in the form of an **X** at the bottom denoting **destruction occurrence**.



Guidelines to draw Sequence Diagrams

- To draw a sequence diagram:
  - List the **actors** and **pre-existing objects** across the top of the diagram;
  - Place the **actor** that initiates the scenario at the **top-left**;
  - Place other actors/objects **from left to right** across the top of the diagram, roughly according to significance;
  - Draw a **vertical dashed line** or **a non dashed line** from the bottom of each actor/object, extending to the bottom of the page. This line is known as a **Life Line**.

Only “significant” details are included because otherwise the diagram will be difficult to read and understand.

Guidelines to draw Sequence Diagrams

- **Time** is shown approximately as distance from the top of the page.

- Messages are placed in order (first to last) from the top of the page to the bottom.

- The **focus of control** is shown by placing solid boxes (executions)over the life lines:
  - the top of the box marks the start of the operation;
  - the bottom of the box marks the end of the operation;
  - the focus of control breaks when a method returns.

Guidelines for Sequence Diagrams

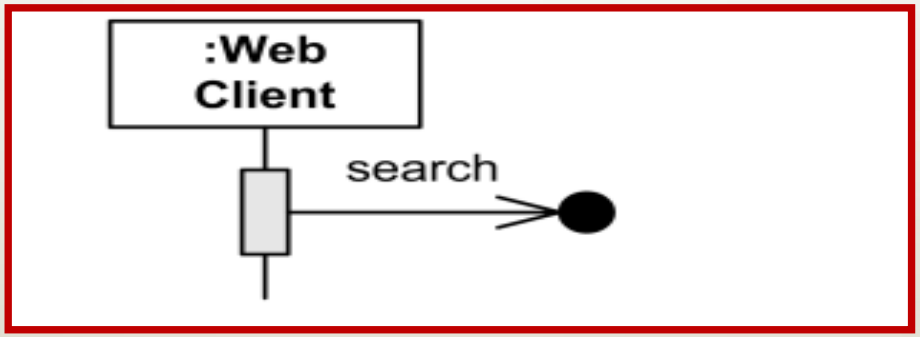
- To draw a Sequence diagram:
  - Examine the **use case scenario** to identify the **actors**.
  - Examine the **class diagram** to identify the **objects**.
  - Draw the actor and class instances.
  - Identify the **messages** needed to realize the scenario:
    - In general, **each step** in the scenario will require **one or more messages** to realize it;
    - An **arrow** is drawn from sending object to the receiving object.

Activity 1

- Draw the partial sequence diagram for the following.
  - The Attendent object is of type AttendanceInterface.
  - The Attendent object requests an anonymous instance of DVDClass to be created.
  - The setTitle message is sent from Attendent to the DVDClass object.
  - The setTitle message sends a boolean value back to the sender (the attendant object), indicating the success (or failure) of the operation.
  - The anonymous DVDClass object is destroyed!

Other Messages - Lost Message

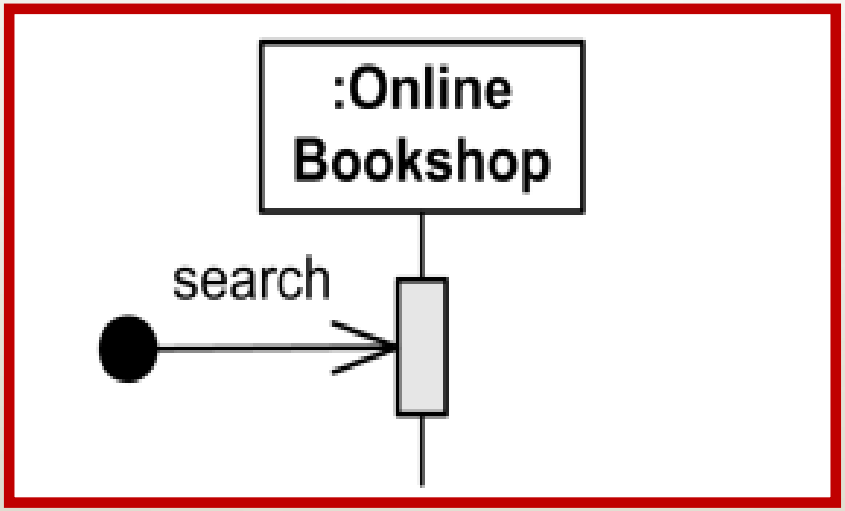
- **Lost Message -** Is a message where the sending event is known, but there is no receiving event. It is interpreted as if the message never reached its destination. The semantics is the trace <sendEvent>, receiveEvent is absent.
- Lost messages are denoted with as a small black circle at the arrow end of the message.



Other Messages - Found Message

- **Found Message** is a message where the receiving event is known, but there is no (known) sending event. It is interpreted as if the origin of the message is outside the scope of the description.

- The semantics is simply the trace: <receiveEvent>, while send event is absent. Found Messages are denoted with a small black circle at the starting end of the message.



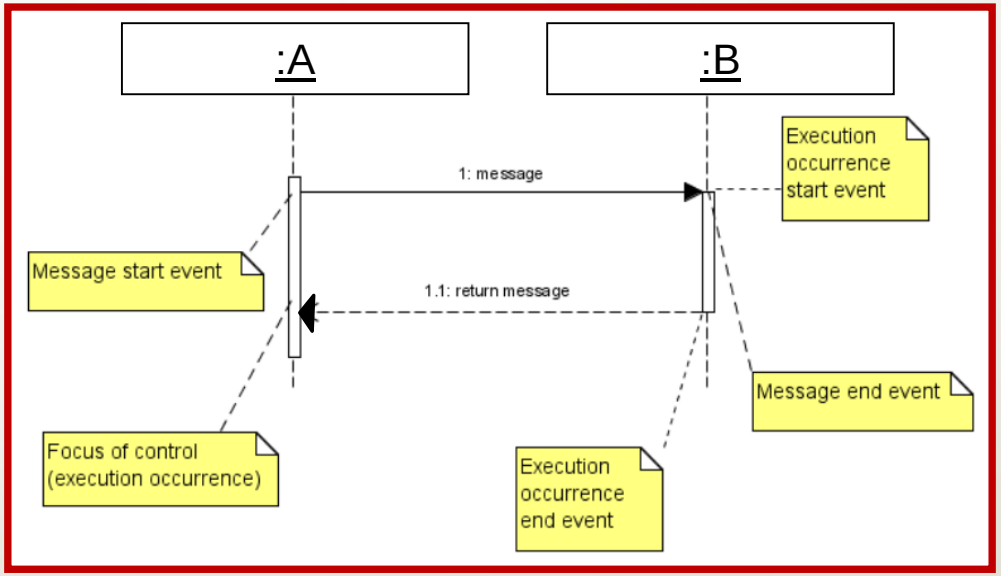
Online Bookshop gets search message of unknown origin

Elements of Sequence Diagram

- Objects
- Life line
- Messages
  - Synchronous call
  - Asynchronous call
  - Create
  - Delete
  - Reply
- Execution

Execution

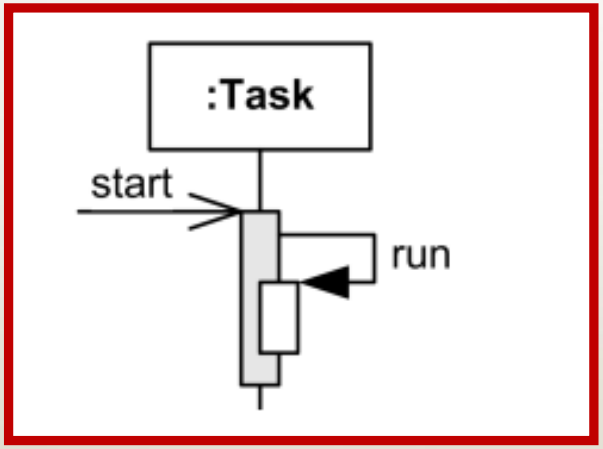
- An operation is **executing** when some process is running its code.
- **Execution** (full name - **execution specification**, informally called **activation**) is **interaction fragment** which represents a period in the participant's lifetime when it is :
  - **executing a unit of behavior or action within the lifeline,**
  - **sending a message to another participant,**
  - **waiting for a reply message from another participant.**



- The **duration** of an execution is represented by two **execution occurrences** - the **start** occurrence and the **finish** occurrence.

Self Calls

- A **self message is** one method calling another method belonging to the same object. (message to the same lifeline)
- It is shown as creating a nested focus of control in the lifeline's execution occurrence.
- Represented by overlapping rectangles on the same lifeline.



Elements of Sequence Diagram

- Objects
- Life line
- Messages
  - Synchronous call
  - Asynchronous call
  - Create
  - Delete
  - Reply
- Execution

Activity 2

The sequence of steps carried out in the "Manage course information" flow are:

A user invokes the *manageCourse* functionality of **Course Administrator**.

The manageCourse function of the **course administrator** class, invokes *courseModification* functionality of a **Course.**

After course is modified, course administrator calls the *topicModification* function of the **Topic**.

Finally, the user invokes *assignTuto*r function of the **Tutor**, to assign a tutor to the selected course.

Draw a sequence diagram for the above given description.

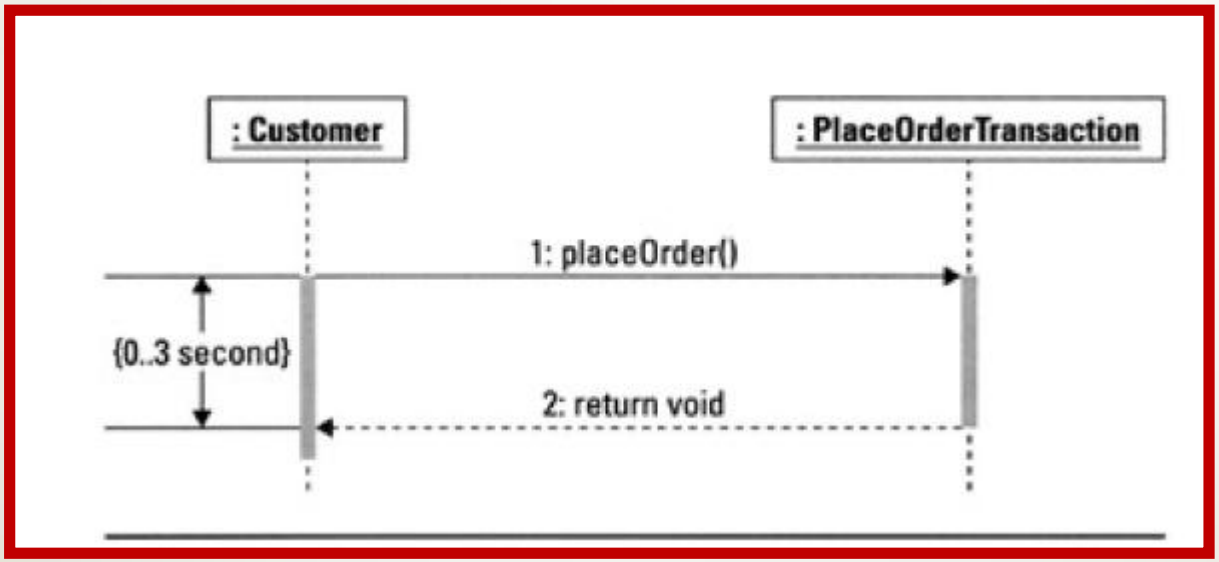
Elements of a Sequence Diagram

- Objects
- Life line
- Messages
  - Synchronous call
  - Asynchronous call
  - Create
  - Delete
  - Reply
- Execution
- Duration and Timing

Duration and Timing

- Sequence diagrams support the ability to specify constraints on the allowed time between events.
  - For example, when a customer submits an order for theater tickets, we need to know that the transaction will not take more than 2 seconds. If it does, we might need to reset the system, recycle the transaction, and/or notify the customer to retry his transaction.
- UML 2.0 provides two types of constraints on the performance characteristics of interactions: duration and time.
  - **Duration :** Refers to the amount of time it takes for something to happen. A duration constraint is a restriction on the length of time it takes to complete a task.
  - **Time :** Refers to a point or period in time when something must happen. A time constraint defines a specific time when a task must be completed.
- E.g. :The duration of the task of writing this page can be specified as between 1 and 2 hours. The time constraint for writing this page may be specified as between Friday at 9:00 AM and Friday at 11:00 AM.

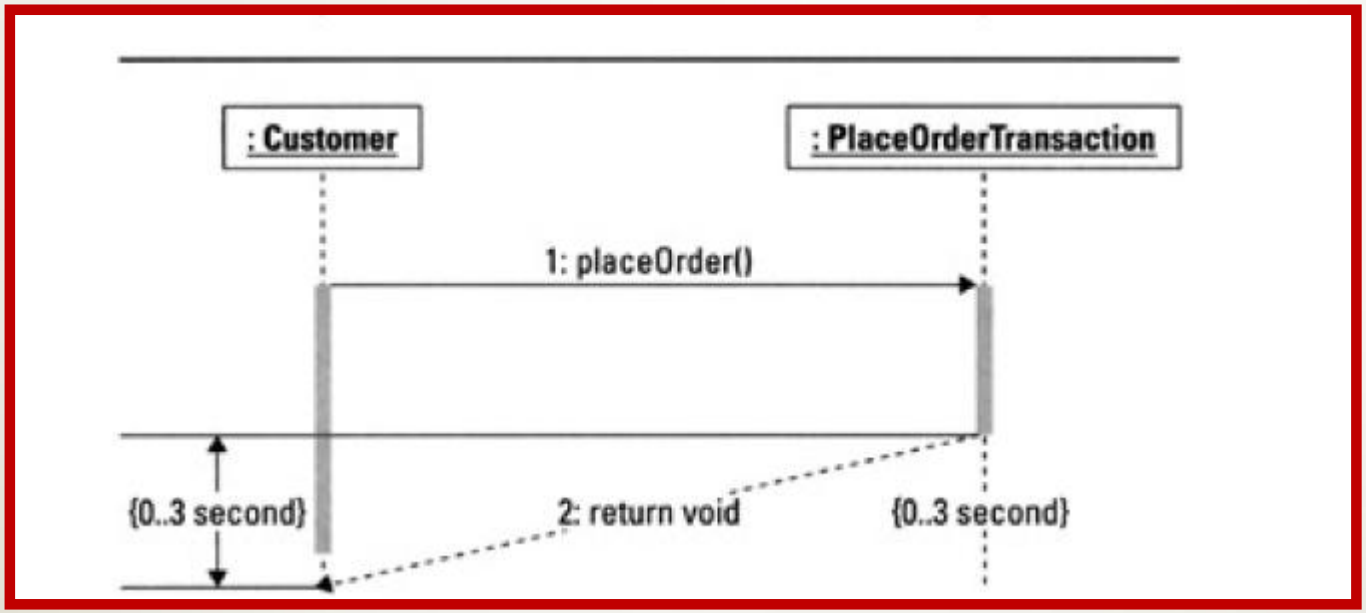
Duration Constraint 01



The figure shows a duration constraint that restricts the amount of time between a sendEvent and a subsequent receiveEvent on the same lifeline.

- The constraint defines a requirement that the application must take between 0 and 3 seconds to go from the sendEvent of the placeOrder() message to the receiveEvent of the return message.

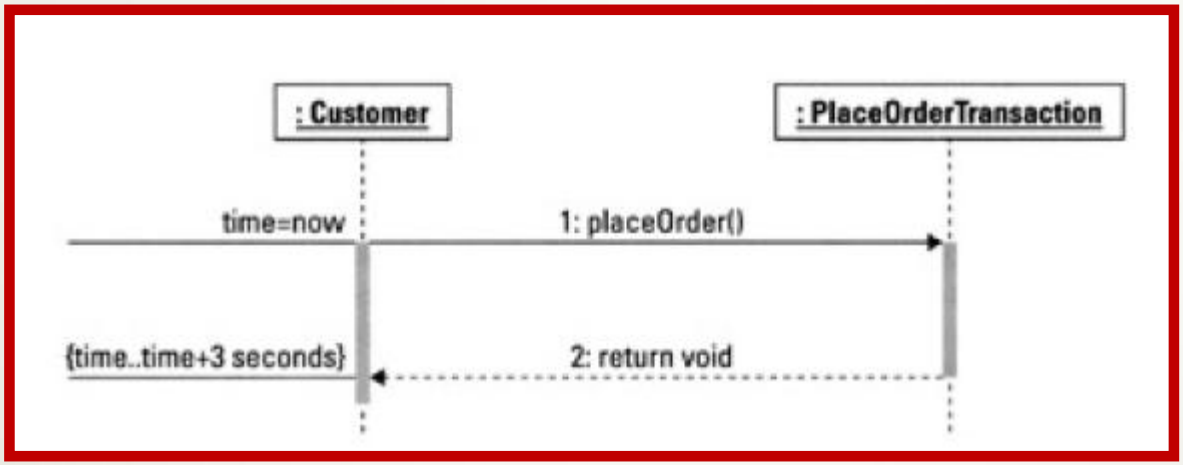
Duration Constraint 02



Duration constraint states that the return from the placeOrder message must be received no more than 3 seconds after the sendEvent of the return.

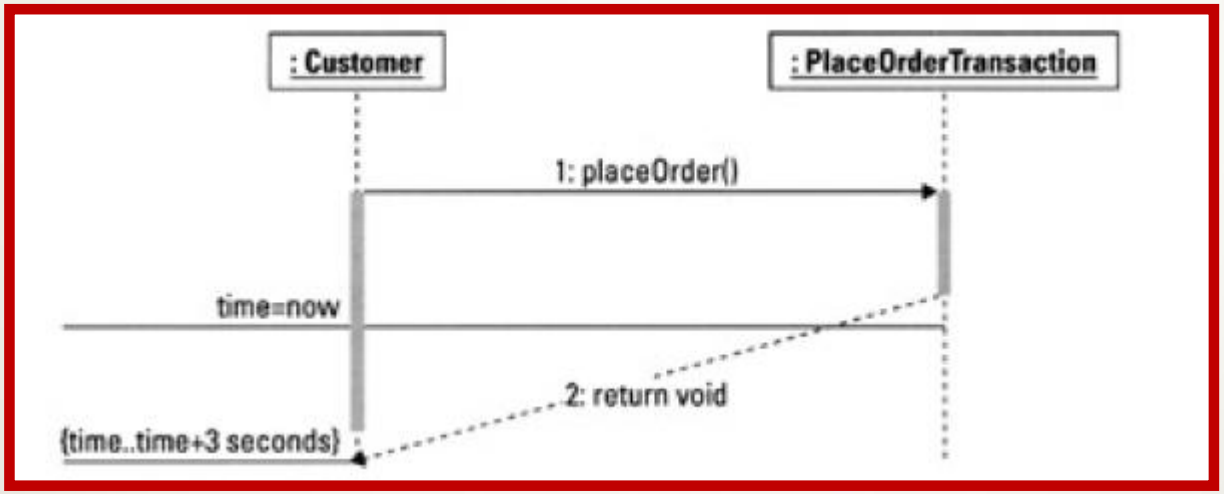
- The duration constraint may be placed along the margin of the diagram (as shown on the left side of the bottom diagram) or as a constraint on the message, in this case message #2 the return.

Time Constraint – representation 1



- Uses a TimeObservationAction (time=now) when Customer sends message #1.
- The time constraint {time..time+3 seconds} requires that message #2 be received no more than 3 seconds after the time established by the TimeObservationAction.

Time Constraint – representation 2



- The time constraint requires message #2 to be received no more than 3 seconds after this event.
- The time event could be a signal, a system event, or a selected point in time such as 3:00 PM.

Activity 3

Draw the sequence diagram for the following scenario

- A day nurse invokes the function *admit(patient)* of **Nurse** class by clicking the Admit Patient button in the system.
- Within the function *admit()*, it calls the *allocateBed()* function of **Bed** class which returns a Bed object.
- To resolve concurrency issues, *allocateBed()* function should be executed within 2 seconds of the function invocation.
- Further, the Bed object should be returned within a maximum of 5 seconds of the *allocateBed()* function invocation.
- Once the bed object is returned, it will be used to create a record by calling *createRecord()* function of **PatientRecord** class.

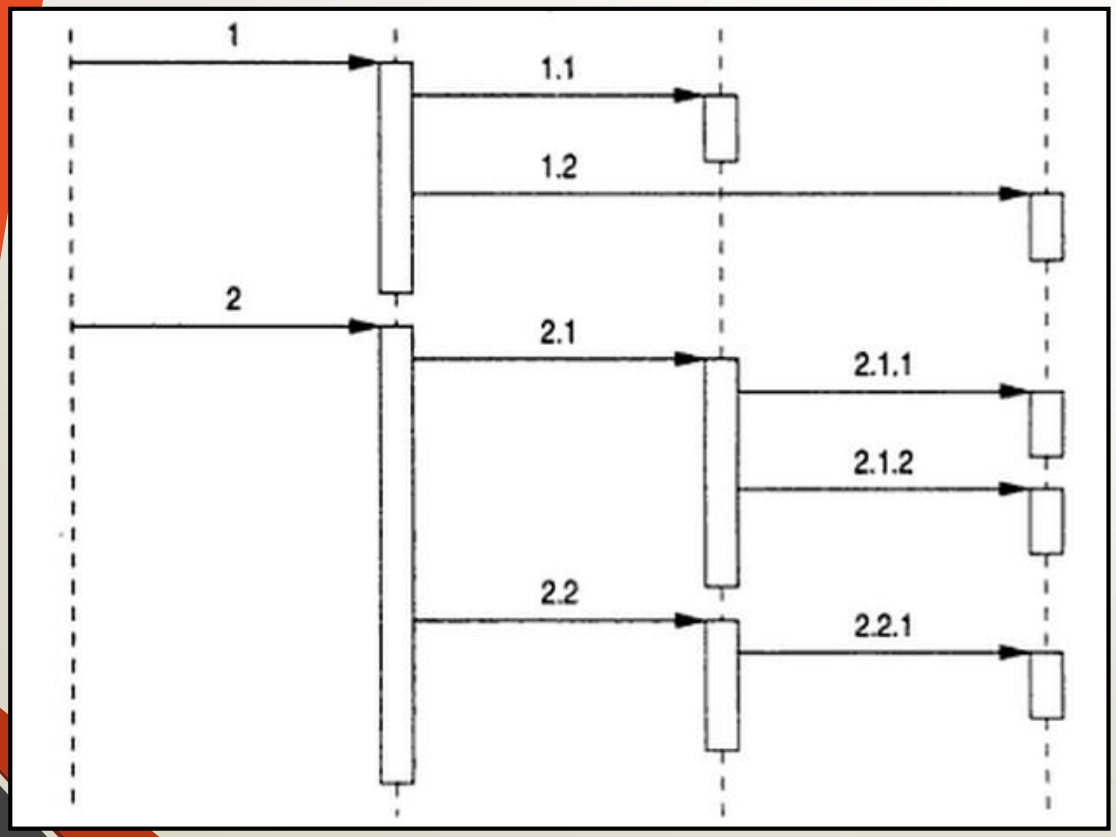
Activity 4– Draw a sequence diagram for the given code

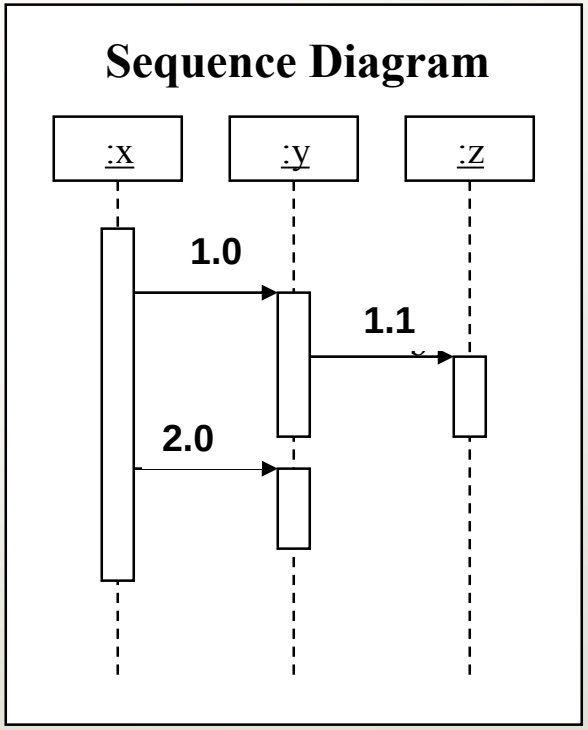
```
Class PrintJobInterface {
      Public static void Print() {
            PrintQueue pq = new PrintQueue();
            Pq.AddPrintJob(doc);
                              }
       Public static void main (string [ ] args ) {
                  PrintJobInterface callprint =new PrintJobInterface ( );
                  callprint.print ( );
                              }
                 }

Class PrintQueue {
       Public void AddPrintJob(Document doc) {
                  Add(doc);
                  ……….
                 PrinterServer ps = new PrinterServer();
                  Ps.Print(doc);
                              }
                 }
Class PrinterServer {
      Public void Print(Document doc) {
                 Printer p = new Printer();
                 p.PrintNow(doc);
                              }
                 }
```

Elements of a Sequence Diagram

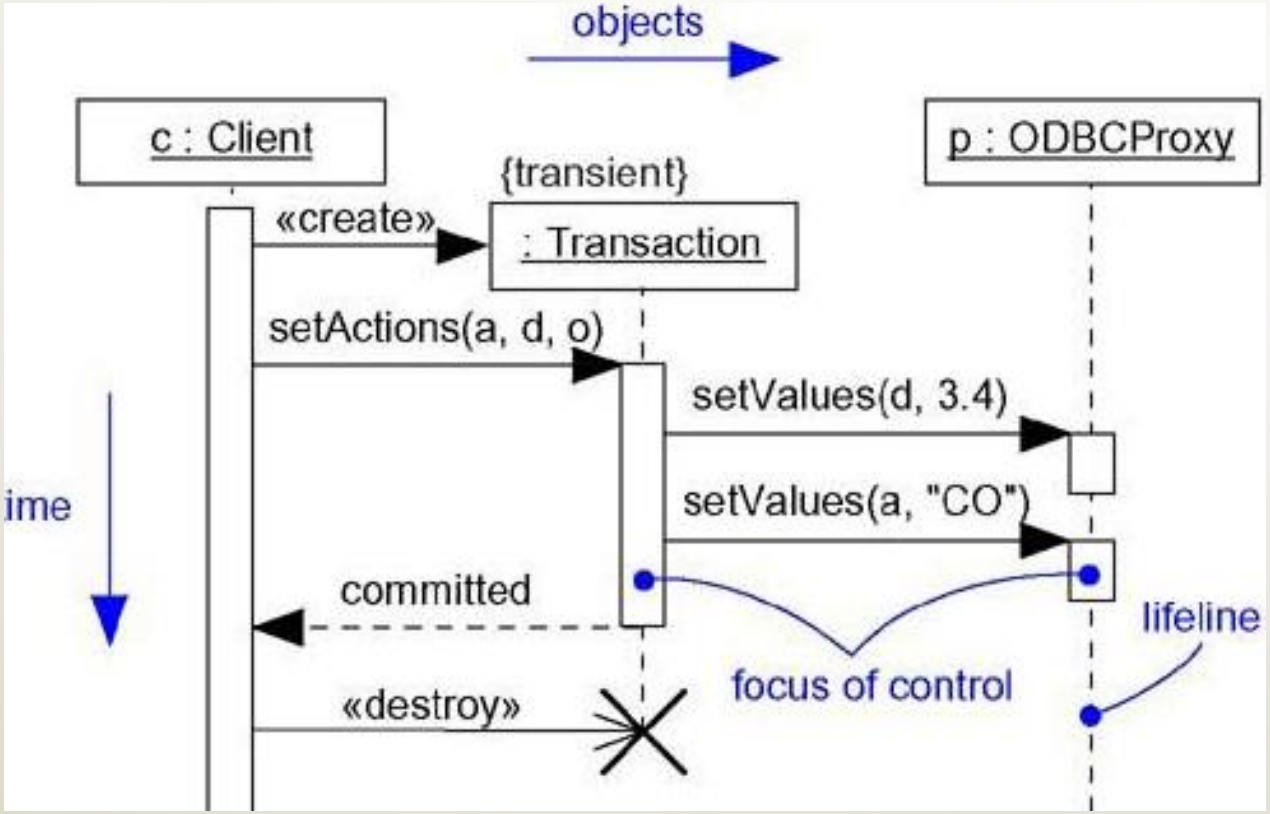
- Objects
- Life line
- Messages
  - Synchronous call
  - Asynchronous call
  - Create
  - Delete
  - Reply
- Execution
- Duration and Timing
- Numbering

Message Numbering





Activity 5- Number the given sequence diagram



Elements of a Sequence Diagram

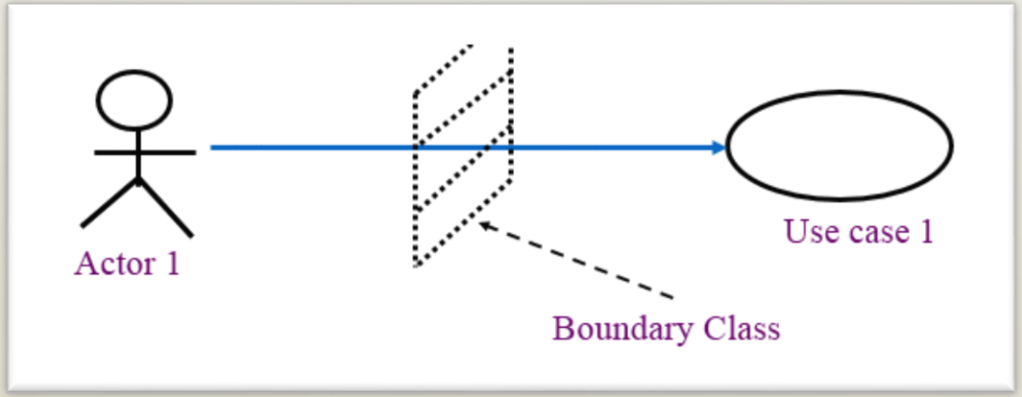
- Objects
- Life line
- Messages
  - Synchronous call
  - Asynchronous call
  - Create
  - Delete
  - Reply
- Execution
- Duration and Timing
- Numbering
- Stereotypes

Stereotypes of Analysis Classes

- Analysis class stereotypes differentiate the roles objects can play.
- Classes or objects can be **classified** into one of the following three stereotypes:
  - Boundary objects model interaction between the system and actors (and other systems)
  - Entity objects represent information and behaviour in the application domain
  - Control objects co-ordinate and control other objects

Boundary Classes

- Boundary is a stereotyped class or object that represents some system boundary, e.g. a user interface screen, system interface .
- It could be used in the analysis or design phase of to capture users or external systems interacting with the system under development.
- It is often used in sequence diagrams which demonstrate user interactions with the system.
- Handles communication between system surroundings and the inside of the system.
- To find the Boundary classes, you can examine your Use Case diagram.
- At a minimum, there must be one *Boundary* class for every actor-use case interaction.



Control Classes

- Control is a stereotyped class or object that is used to model flow of control or some coordination in behavior.
- Co-ordinates the events needed to realize the behaviour specified in the use case.
- Models complex computations and algorithms.
- There is at least *one control class per use case*

Eg. Running or *executing* the use case.

- One or several control classes could describe use case realization.
- System controls represent the dynamics of the designed system and usually describe some "business logic".

Entity Class

Entity is a stereotyped class or object that represents some information or data.

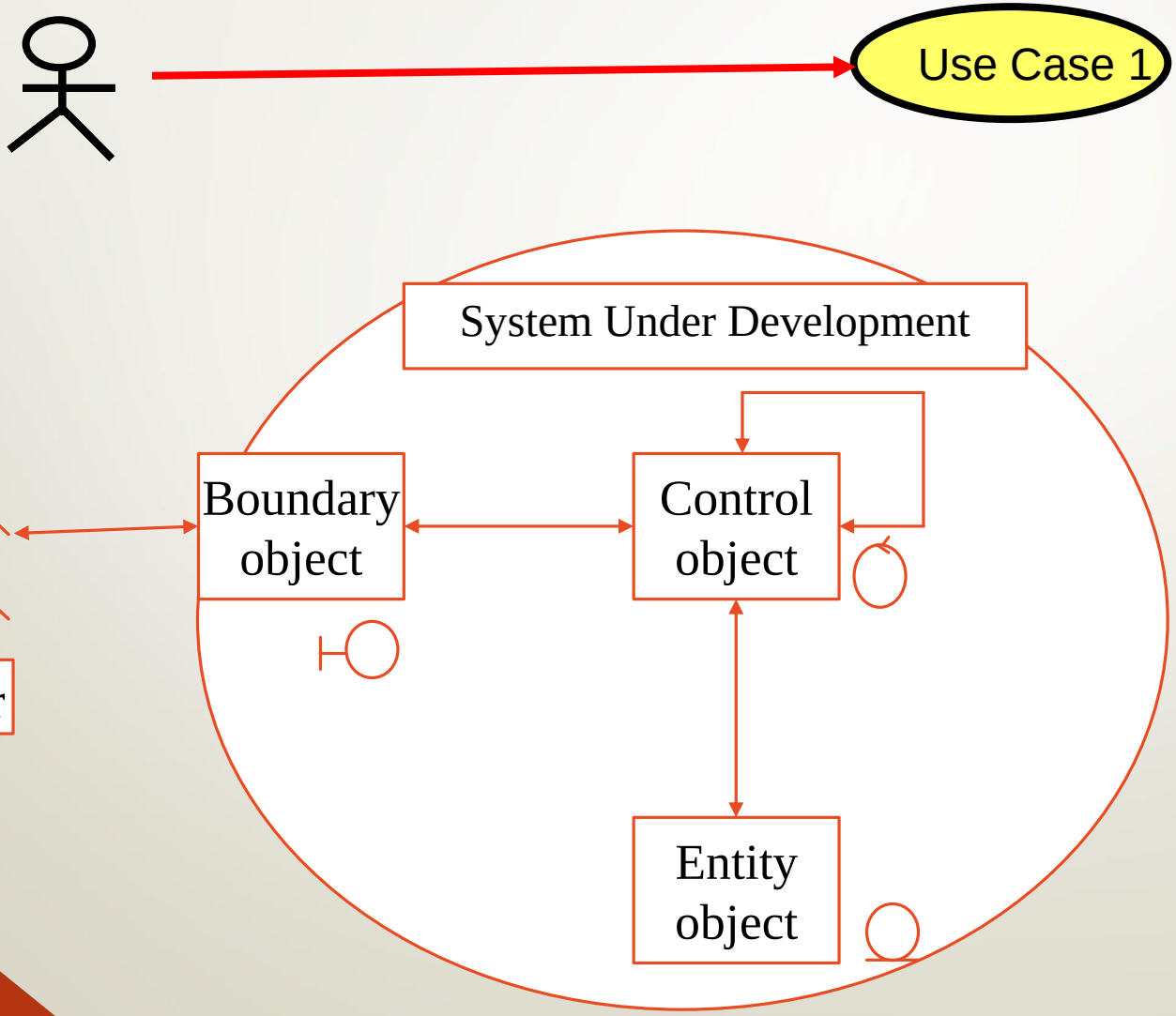
These classes are needed to perform tasks internal to the system.

It reflects a real world entity.

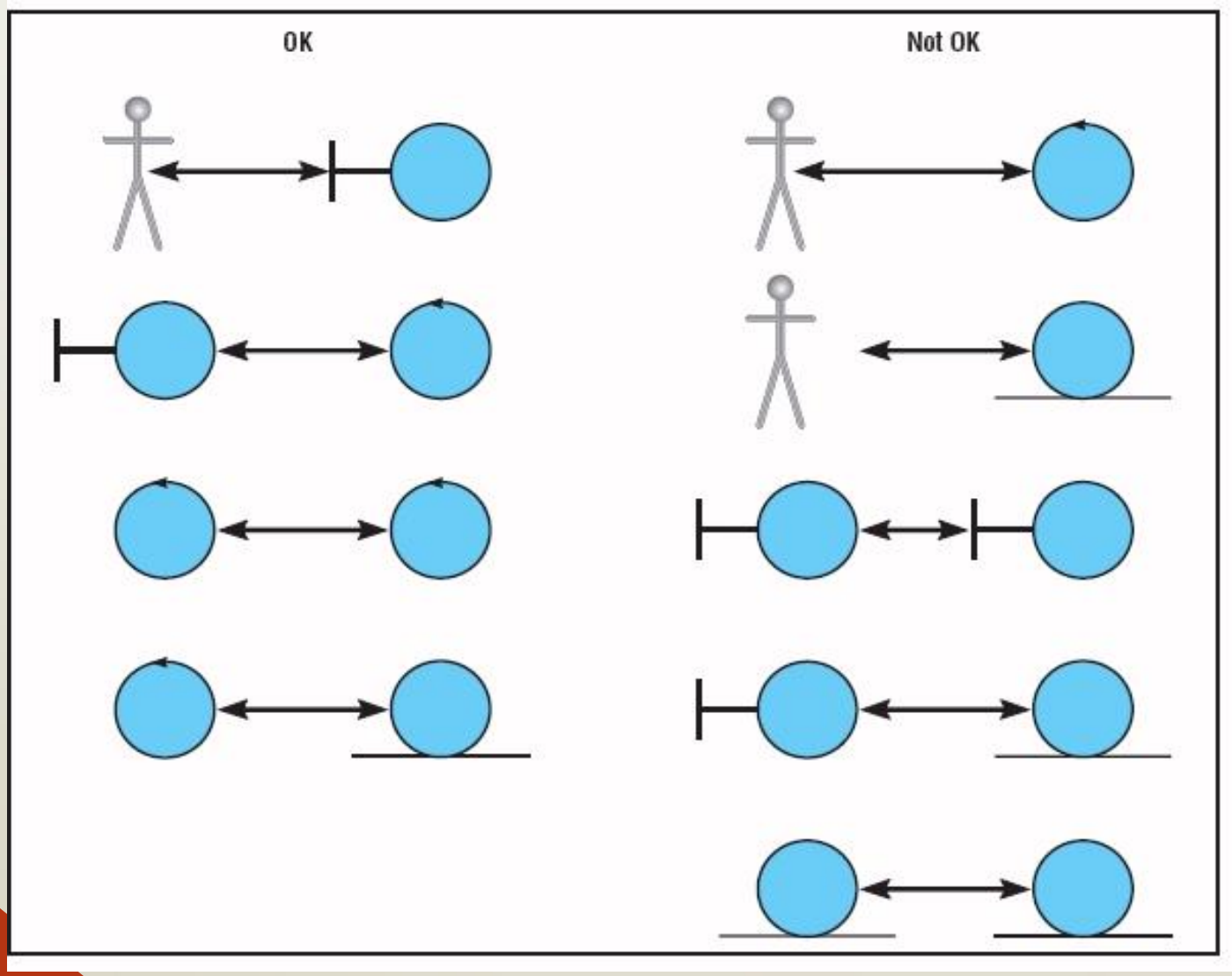
Eg:

- *Account*
- *Student*
- *Lecturer*, etc…

Block Diagram:



Rules of Sequence Diagrams and Stereotypes



Activity 6– Draw a sequence diagram for the given "Inquire book status" scenario by identifying stereotypes.

Name: Inquire book status

Brief Description: Handle from the borrower about availability of a book.

Actor: Librarian

Flow of Events:

1. Librarian clicks " showBookStatus" in the user interface by giving the item ID.

2. Then system will internally call "getItemList" in the Item class to get relevant details.

3. Then Item details are displayed by the system for the Librarian.

Interaction Diagrams

- Interaction diagrams provide a graphical representation of a runtime environment, composed from:
  - instances of **actors -** typically anonymous;
  - instances of **classes** - i.e. objects!
  - **messages** (i.e. method calls and returns) between the above instances.
- Interaction diagrams DO NOT describe the low level details of an algorithm:
  - they DO describe the **message passing** required by an algorithm that **realizes a use case**.

References

- UML 2 Bible
  - Chapters 8 & 9
- Applying UML and Patterns by Craig Larman
  - Chapter 15