

# CS1007: Object Oriented Design and Programming in Java

Lecture #5

Sept 20

Shlomo Hershkop  
*shlomo@cs.columbia.edu*

## Outline

- Feedback
- More UML and OOD
  
- Reading
  - Chapter 2
- Next
  - Chapter 3

## Feedback

- Lots of UML confusion
- Class design issues
- Why cs accounts
  - Will need it for lab class (to be announced)
  - Allow you to gain extra skills outside of basic 1007








## UML

- Why do we model?
  - Provide structure for problem solving
  - Experiment to explore multiple solutions
  - Furnish abstractions to manage complexity
  - Decrease development costs
  - Manage the risk of mistakes
- Graphical Approach
  - Picture is worth 1000 words

# UML Building Blocks

- model elements (classes, interfaces, components, use cases, etc.)
- relationships (associations, generalization, dependencies, etc.)
- diagrams (class diagrams, use case diagrams, interaction diagrams, etc.)
- Simple building blocks are used to create large, complex structures
  - elements, bonds and molecules in chemistry
  - components, connectors and circuit boards in hardware

# UML Relationships

Dependency		1. Proper operation of one depends on another
Aggregation		2. Has-a part-whole <ul style="list-style-type: none"><li>1. Student-&gt;department</li></ul>
Inheritance		2. Faculty member->dept <ul style="list-style-type: none"><li>3. Dept -&gt;college</li></ul>
Composition		3. Is-a
Association		4. One doesn't exist without the other
Directed Association		
Interface Type Implementation		

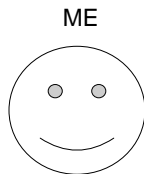
## Java definitions

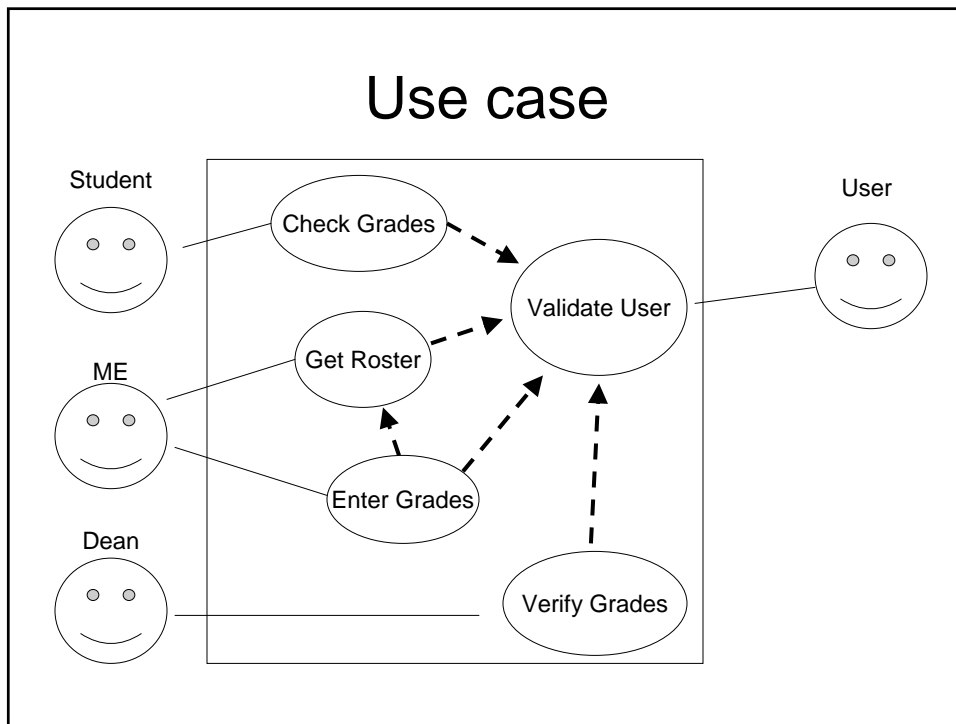
- When class X extends Y
  - X is a subclass
  - Y is a superclass
- When interface A extends Interface B
  - A is a subinterface
  - B is a superinterface
- When G implements interface B
  - G is an implementation of B
  - B is an interface of class G

- Class diagrams – static relationships
- Dynamic relationship – interaction between objects and ordering of events
  - Sequence – time order layout of events
  - State diagrams – flow of control
    - FSM – Finite State Machines
    - Event – is a occurrence that triggers a state transition
    - Nested – diagrams can contain other diagrams

## Use cases

- Consists of interactions between the system and “actors” and their relationships
- Describes what the system does (not how)
- High level sketch





## From Last Class

- Middle of documenting a voice mail system

## Voice Mail System

- Use text for voice, phone keys, hangup
- 1 2 ... 0 # on a single line means key
- H on a single line means "hang up"
- All other inputs mean voice
- In GUI program, will use buttons for keys (see ch. 4)

## Reach an Extension

1. User dials main number of system
2. System speaks prompt

Enter mailbox number followed by #

3. User types extension number
4. System speaks

You have reached mailbox xxxx.  
Please leave a message now

## Leave a Message

1. Caller carries out Reach an Extension
2. Caller speaks message
3. Caller hangs up
4. System places message in mailbox

## Log In

1. Mailbox owner carries out Reach an Extension
2. Mailbox owner types password and # (Default password = mailbox number. To change, see Change the Passcode)
3. System plays mailbox menu:

Enter 1 to retrieve your messages.

Enter 2 to change your passcode.

Enter 3 to change your greeting.



## Retrieve Messages

1. Mailbox owner carries out Log in
2. Mailbox owner selects "retrieve messages" menu option
3. System plays message menu:

Press 1 to listen to the current message

Press 2 to delete the current message

Press 3 to save the current message

Press 4 to return to the mailbox menu

4. Mailbox owner selects "listen to current message"
5. System plays current new message, or, if no more new messages, current old message.
6. Note: Message is played, not removed from queue
7. System plays message menu
8. User selects "delete current message". Message is removed.
9. Continue with step 3.

## Change Greeting

1. Mailbox owner carries out Log in
2. Mailbox owner selects "change greeting" menu option
3. Mailbox owner speaks new greeting
4. Mailbox owner presses #
5. System sets new greeting

## Change Passcode

- Mailbox owner carries out Log in
- Mailbox owner selects "change passcode" menu option
- Mailbox owner dials new passcode
- Mailbox owner presses #
- System sets new passcode

## CRC Cards: Mailbox

Mailbox	
<i>keep new and saved messages</i>	MessageQueue

## CRC Cards: MessageQueue

MessageQueue	
<i>add and remove messages in FIFO order</i>	

## CRC Cards: MailSystem

MailSystem	
<i>manage mailboxes</i>	Mailbox

# Telephone

- Who interacts with user?
- Telephone takes button presses, voice input
- Telephone speaks output to user

Telephone
<i>take user input from touchpad,</i>
<i>microphone, hangup</i>
<i>speak output</i>

## Connection

- With whom does Telephone communicate
- With MailSystem?
- What if there are multiple telephones?
- Each connection can be in different state
  - (dialing, recording, retrieving messages,...)
- Should mail system keep track of all connection states?
- Better to give this responsibility to a new class

## Connection

Connection	
<i>get input from telephone</i>	Telephone
<i>carry out user commands</i>	MailSystem
<i>keep track of state</i>	

## Analysis: Leave Message

1. User dials extension. Telephone sends number to Connection  
(Add collaborator Telephone to Connection)
2. Connection asks MailSystem to find matching Mailbox
3. Connection asks Mailbox for greeting  
(Add responsibility "manage greeting" to Mailbox,  
add collaborator Mailbox to Connection)
4. Connection asks Telephone to play greeting
5. User speaks greeting. Telephone asks Connection to record it.  
(Add responsibility "record voice input" to Connection)
6. User hangs up. Telephone notifies Connection.
7. Connection constructs Message  
(Add card for Message class,  
add collaborator Message to Connection)
8. Connection adds Message to Mailbox

Telephone	
<i>take user input from touchpad,</i>	Connection
<i>microphone, hangup</i>	
<i>speak output</i>	

Connection	
<i>get input from telephone</i>	Telephone
<i>carry out user commands</i>	MailSystem
<i>keep track of state</i>	Mailbox
<i>record voice input</i>	Message

Mailbox	
<i>keep new and saved messages</i>	MessageQueue
<i>manage greeting</i>	



## Retrieve Messages

1. User types in passcode. Telephone notifies Connection
2. Connection asks Mailbox to check passcode.  
(Add responsibility "manage passcode" to Mailbox)
3. Connection sets current mailbox and asks Telephone to speak menu
4. User selects "retrieve messages". Telephone passes key to Connection
5. Connection asks Telephone to speak menu
6. User selects "listen to current message". Telephone passes key to Connection
7. Connection gets first message from current mailbox.  
(Add "retrieve messages" to responsibility of Mailbox).  
Connection asks Telephone to speak message
8. Connection asks Telephone to speak menu
9. User selects "save current message". Telephone passes key to Connection
10. Connection tells Mailbox to save message  
(Modify responsibility of Mailbox to "retrieve,save,delete messages")
11. Connection asks Telephone to speak menu



## Result of Use Case Analysis

Mailbox	
<i>keep new and saved messages</i>	MessageQueue
<i>manage greeting</i>	
<i>manage passcode</i>	
<i>retrieve, save, delete messages</i>	

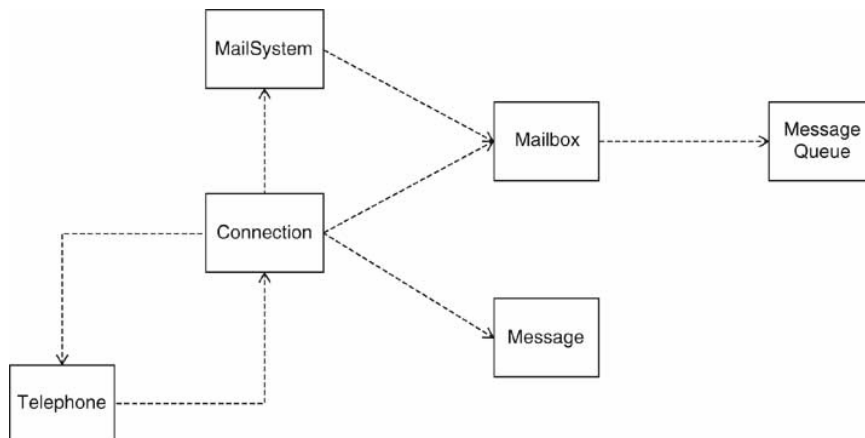
## CRC Summary

- One card per class
- Responsibilities at high level
- Use scenario walkthroughs to fill in cards
- Usually, the first design isn't perfect.
  - (You just saw the author's third design of the mail system)

## UML Class Diagram for Mail System

- CRC collaborators yield dependencies
- Mailbox depends on MessageQueue
- Message doesn't depends on Mailbox
- Connection depends on Telephone, MailSystem, Message, Mailbox
- Telephone depends on Connection

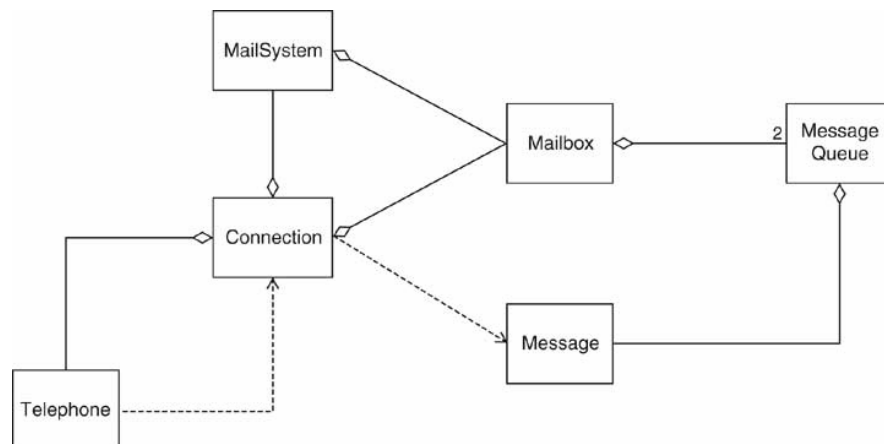
## Dependency Relationships



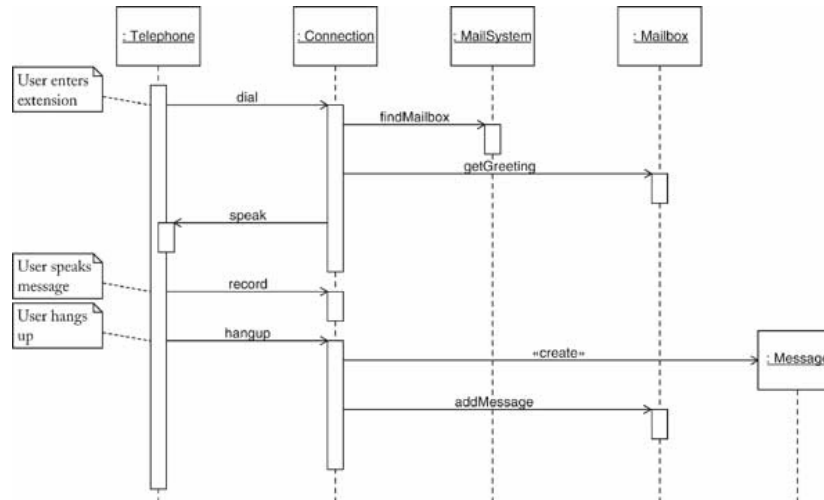
## Aggregation Relationships

- A mail system has mailboxes
- A mailbox has two message queues
- A message queue has some number of messages
- A connection has a current mailbox.
- A connection has references to a mailsystem and a telephone

## UML Class Diagram for Voice Mail System



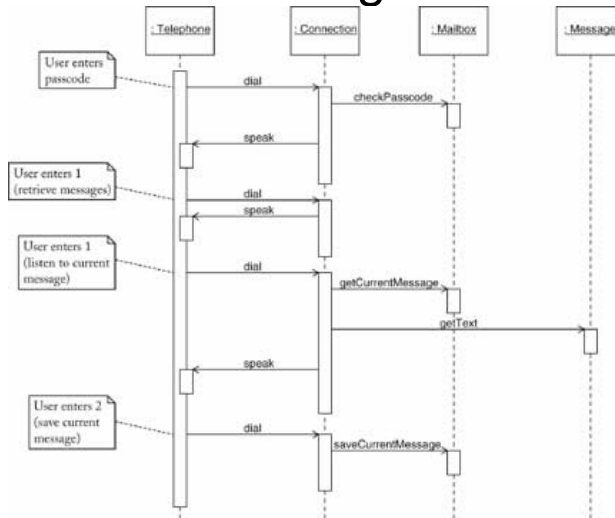
## Sequence Diagram for Use Case: Leave a message



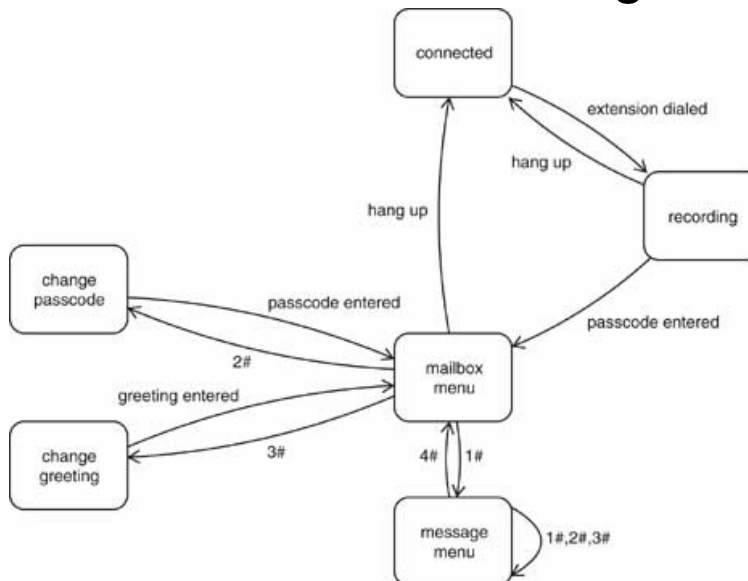
## Interpreting a Sequence Diagram

- Each key press results in separate call to dial, but only one is shown
- Connection wants to get greeting to play
- Each mailbox knows its greeting
- Connection must find mailbox object:  
Call findMailbox on MailSystem object
- Parameters are not displayed (e.g. mailbox number)
- Return values are not displayed (e.g. found mailbox)
- Note that connection holds on to that mailbox over multiple calls

## Sequence Diagram: Retrieve messages



## Connection State Diagram



## Java Example

```
01: /**
02:  A message left by the caller.
03: */
04: public class Message
05: {
06:     /**
07:      Construct a Message object.
08:      @param messageText the message text
09:     */
10:     public Message(String messageText)
11:     {
12:         text = messageText;
13:     }
14:
15:     /**
16:      Get the message text.
17:      @return message text
18:     */
19:     public String getText()
20:     {
21:         return text;
22:     }
23:
24:     private String text;
25: }
```

## For MessageQueue

```
36:     /**
37:      Get the total number of messages in the queue.
38:      @return the total number of messages in the queue
39:     */
40:     public int size()
41:     {
42:         return queue.size();
43:     }
44:
45:     /**
46:      Get message at head.
47:      @return message that is at the head of the queue, or null
48:      if the queue is empty
49:     */
50:     public Message peek()
51:     {
52:         if (queue.size() == 0) return null;
53:         else return queue.get(0);
54:     }
55:
56:     private ArrayList<Message> queue;
57: }
```

# Tester

```
01: import java.util.Scanner;
02:
03: /**
04:     This program tests the mail system. A single phone
05:     communicates with the program through
06:     System.in/System.out.
07: */
08: public class MailSystemTester
09: {
10:     public static void main(String[] args)
11:     {
12:         MailSystem system = new MailSystem(MAILBOX_COUNT);
13:         Scanner console = new Scanner(System.in);
14:         Telephone p = new Telephone(console);
15:         Connection c = new Connection(system, p);
16:         p.run(c);
17:     }
18:     private static final int MAILBOX_COUNT = 20;
19: }
```

# Voilet

- Demo of UML system

## Next Time

- Considerations when choosing and designing classes