The Object-Oriented Design Process

Horstmann Section 2.12

CS 3354
Spring 2017

Jill Seaman

2.12 Case Study: A Voice Mail System

Consider the task of writing a program to simulate the following simple telephone voice mail system:

- (Assuming the caller has already dialed into the system)
  A person dials an extension number and, provided the other party does not pick up the telephone, receives a greeting and leaves a message. The other party can later retrieve their messages, keep them, or delete them, once they have provided their passcode.
- We will use text to represent voice, phone keys, hanging up:
  - ✦ 1 2 ... 0 # on a single line denotes the corresponding phone key
  - ✦ H on a single line means "hang up"
  - ✦ All other inputs mean voice
- We will begin with the analysis phase and develop use cases.

2.12.1 Use cases for the Voice Mail System
Use Case: Reach an Extension

Reach an Extension:

1. The caller dials the main number of the voice mail system.
2. The voice mail system speaks the following prompt:

   ```
   Enter mailbox number followed by #.
   ```

3. The caller types in the extension number of the message recipient.
4. The voice mail system speaks the following message:

   ```
   You have reached mailbox xxxx. Please leave a message now.
   ```


Use Case: Leave a Message

Leave a Message

1. The caller carries out **Reach an Extension**.
2. The caller speaks the message.
3. The caller hangs up.
4. The voice mail system places the recorded message in the recipient's mailbox.

Use Case: Log in

1. The mailbox owner carries out **Reach an Extension**.
2. The mailbox owner types the passcode and the # key
   (Default passcode = mailbox number.
   To change the passcode, see **Change the Passcode**)
3. The voice mail system plays the mailbox menu:

```
Enter 1 to retrieve your messages.
Enter 2 to change your passcode.
Enter 3 to change your greeting.
```


Use Case: Retrieve Messages

1. The mailbox owner carries out **Log in**.
2. The mailbox owner selects option 1 retrieve your messages.
3. The voice mail system plays the message menu:

```
Enter 1 to listen to the current message.
Enter 2 to save the current message.
Enter 3 to delete the current message.
Enter 4 to return to the mailbox menu.
```

4. The mailbox owner selects option 1 listen to the current message.
5. The voice mail system plays the current new message, or, if there are no new messages, the current old message. Note that the message is played, not removed from the queue.
6. The voice mail system plays the message menu.
7. The user selects option 3 delete the current message. The message is permanently removed.
8. Continue with Step 3.

Use Case: Retrieve Messages - Variations

Variation #1 Saving a message

1.1. Start at Step 6

1.2. The user selects option 2 save the current message. The message is removed from its queue and appended to the queue of old messages.

1.3. Continue with Step 3.

Use Case: Change the Greeting

Change the Greeting

1. The mailbox owner carries out **Log in**.
2. The mailbox owner selects option 3 change your greeting.
3. The mailbox owner speaks the greeting.
4. The mailbox owner presses the # key.
5. The mail system sets the new greeting.

Variation #1. Hang up before confirmation

1.1. Start at Step 3.

1.2. The mailbox owner hangs up the telephone.

1.3. The mail system keeps the old greeting.

Use Case: Change the Passcode

Change the Passcode

1. The mailbox owner carries out **Log in**.
2. The mailbox owner selects option 2 change your passcode.
3. The mailbox owner dials the new passcode
4. The mailbox owner presses the # key.
5. The mail system sets the new passcode.

Variation #1. Hang up before confirmation

1.1. Start at Step 3.

1.2. The mailbox owner hangs up the telephone.

1.3. The mail system keeps the old passcode.

2.12.2 CRC Cards for the Voice Mail System

- Next let's discover the classes for the system.
- These are obvious
  - ✦ Mailbox
  - ✦ Message
  - ✦ MailSystem
- **Mailbox** responsibilities:
  - ✦ keep messages
  - ✦ which are new, which are saved.
  - ✦ users should be able to retrieve, save, and delete messages
  - ✦ need a queue to store the messages in FIFO order: MessageQueue

Initial Mailbox CRC Card

- Remember: responsibilities on the left, collaborating classes on the right.

- Class=Mailbox
- Responsibilities:
  - ✦keep new and saved messages
- Collaborators:
  - ✦MessageQueue

Mailbox	
keep new and saved messages	MessageQueue

Initial MessageQueue CRC Card

- Class=MessageQueue
- Responsibilities:
  - ✦add and remove messages in FIFO order
- Collaborators:
  - ✦Message

MessageQueue	
add and remove messages in FIFO order	Message

Initial MailSystem CRC Card

- We need a class to contain (and find) the Mailboxes

- Class=MailSystem
- Responsibilities:
  - ✦manage mailboxes
- Collaborators:
  - ✦Mailbox

MailSystem	
manage mailboxes	Mailbox

Initial Telephone CRC Card

- We need a class to manage input and output

- Class=Telephone
- Responsibilities:
  - ✦take user input from touchpad, microphone, hangup
  - ✦speak output
- Collaborators:
  - ✦None

Telephone
take user input from touchpad, microphone, hangup
speak output

Initial Connection CRC Card

- We need a class to manage the connection between a telephone and the MailSystem
- The telephone could do this, but it would have too much responsibility.
- Class=Connection
- Responsibilities:
  - ✦get input from telephone
  - ✦carry out user commands
  - ✦keep track of state
- Collaborators:
  - ✦Telephone
  - ✦MailSystem

Connection	
get input from telephone	Telephone
carry out user commands	MailSystem
keep track of state	

CRC Card process: Walkthrough Leave a Message

1 User dials extension. `Telephone` sends number to `Connection`
(Add collaborator `Connection` to `Telephone`)
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 message. `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 (we already did this),
add collaborator `Message` to `Connection`)
8 `Connection` adds `Message` to `Mailbox`

Changed/Added CRC Cards

Telephone	
take user input from touchpad, microphone, hangup	Connection
speak output	

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

Mailbox	
keep new and saved messages	MessageQueue
manage greeting	

Message
manage message contents

Changed/Added CRC Cards

- Class=Connection
- Responsibilities:
  - ✦get input from telephone
  - ✦carry out user commands
  - ✦keep track of state
  - ✦record voice input
- Collaborators:
  - ✦Telephone
  - ✦MailSystem
  - ✦Mailbox
  - ✦Message

- Class=Telephone
- Responsibilities:
  - ✦take user input from touchpad, microphone, hangup
  - ✦speak output
- Collaborators:
  - ✦Connection

- Class=Mailbox
- Responsibilities:
  - ✦keep new and saved messages
  - ✦manage greeting
- Collaborators:
  - ✦MessageQueue

- Class=Message
- Responsibilities:
  - ✦manage message contents
- Collaborators:
  - ✦None

CRC Card process: Walkthrough 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`).
8 `Connection` asks `Telephone` to speak message
9 `Connection` asks `Telephone` to speak menu
10 User selects "save current message". `Telephone` passes key to `Connection`
11 `Connection` tells `Mailbox` to save message
(Modify responsibility of `Mailbox` to "retrieve,save,delete messages")
12 `Connection` asks `Telephone` to speak menu

Updated Mailbox CRC Card

- New responsibilities assigned after Retrieve Messages walkthrough:
- Class=Mailbox
- Responsibilities:
  - ✦keep new and saved messages
  - ✦manage greeting
  - ✦manage passcode
  - ✦retrieve, save, delete messages
- Collaborators:
  - ✦MessageQueue

Mailbox	
keep new and saved messages	MessageQueue
manage greeting	
manage passcode	
retrieve, save, delete messages	

CRC Cards process

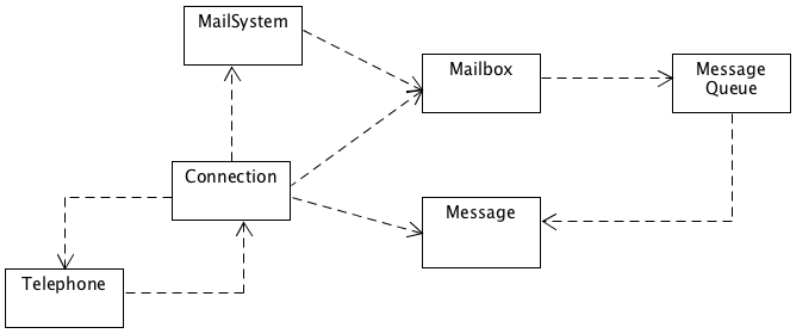
- Use one card per class
- State responsibilities at high level
- Use scenario walkthroughs to fill in cards, make new ones, move responsibilities from one card to another, delete unneeded cards.
- Usually, the first design isn't perfect.
- The process should be iterative.
- (You just saw the author's third design of the mail system)

2.12.3 Class diagrams for the Voice Mail System

CRC card collaborators yield dependencies:

- Mailbox depends on MessageQueue
- MailSystem depends on MailBox
- Connection depends on Telephone, MailSystem, Message, Mailbox
- Telephone depends on Connection
- Message doesn't depend on anything
- MessageQueue depends on Message

The Voice Mail System Dependencies from the CRC Cards



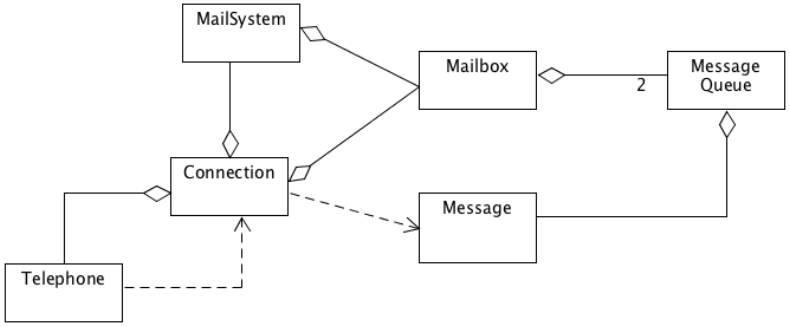
Class diagrams for the Voice Mail System

Next, consider the aggregation relationships:

- A MailSystem 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

The Voice Mail System from the CRC Cards

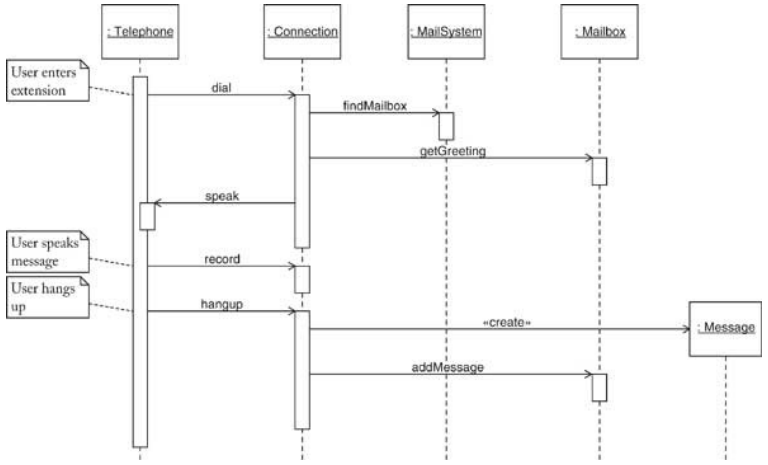
- There is no inheritance among these classes
- Aggregation associations replace pre-existing dependency relationships.



2.12.4 UML Sequence and State Diagrams

- The purpose of a sequence diagram is to understand a complex control flow that involves multiple objects.
- A sequence diagram usually represents the steps of a use case.
- They are not needed for simple interactions.
- For the Voice Mail System, the interactions between the Telephone, Connection, MailSystem, and Mailbox classes are not easy to understand.
- So we'll draw a sequence diagram for Leave a Message.

Sequence Diagram for Leave a Message



Sequence Diagram: Leave a Message

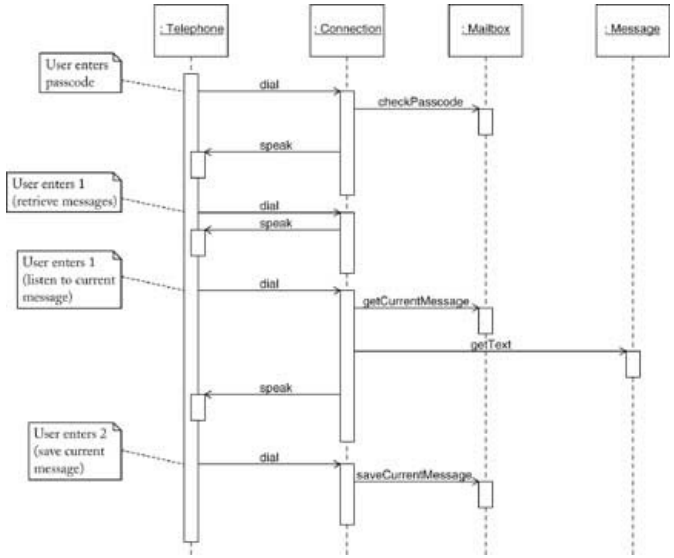
- During this process, we decide what methods the classes need.
- Connection needs three methods to receive data from the Telephone:
  - `dial` passes on a button press.
  - `record` passes on (some) speech.
  - `hangup` tells the connection that the telephone has hung up.
- Connection needs to get greeting to play
  - Each mailbox knows its greeting
  - Connection must find mailbox object:
  - Call `findMailbox` on MailSystem object, then call `getGreeting` on that Mailbox
  - Call `speak` on the Telephone to play the greeting.
- Telephone records the (next line of the) message, and passes it to the Connection by calling `record`.
- Telephone calls `hangup`.

Sequence Diagram: Leave a Message

- Connection must construct a Message and add it to the mailbox (`addMessage`)
  - which mailbox object?
  - the one from the previous call to `findMailbox`

Note:
- Each key press results in separate call to `dial`, but only one is shown
- 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 for Retrieve Messages

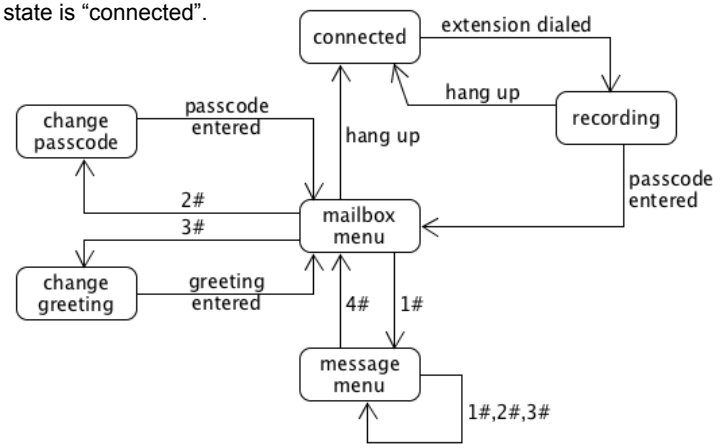


State diagrams

- Input is not controlled by the voice mail system
  - ✦It responds to input from the Telephone
- The Connection needs to keep track of what state it is in so it knows how to respond to the given input.

State Diagram of the Connection Class

The starting state is “connected”.



2.12.5 Java Implementation

- Compare the Java code to the CRC cards, the class diagram, the sequence diagrams, and the state diagram
- A link to the code is on the class website/TRACS (mail.zip)