CS1007: Object Oriented Design and Programming in Java

Lecture #4

Sept 15

Shlomo Hershkop
shlomo@cs.columbia.edu

Outline

- Feedback
- Object Oriented Design Process.
- CRC
- UML
- Example: VoiceMail System

- Reading chapter 2-2.5
  Next: 2.5-

Announcements

- Please make sure to note : HW due: sept 27
- Bill Gates will be visiting on Oct 13, tickets will be made available if you are interested (fcfs).
- Beginning Monday, I will try to post weekly slides for the week (ahead of classes).

Feedback

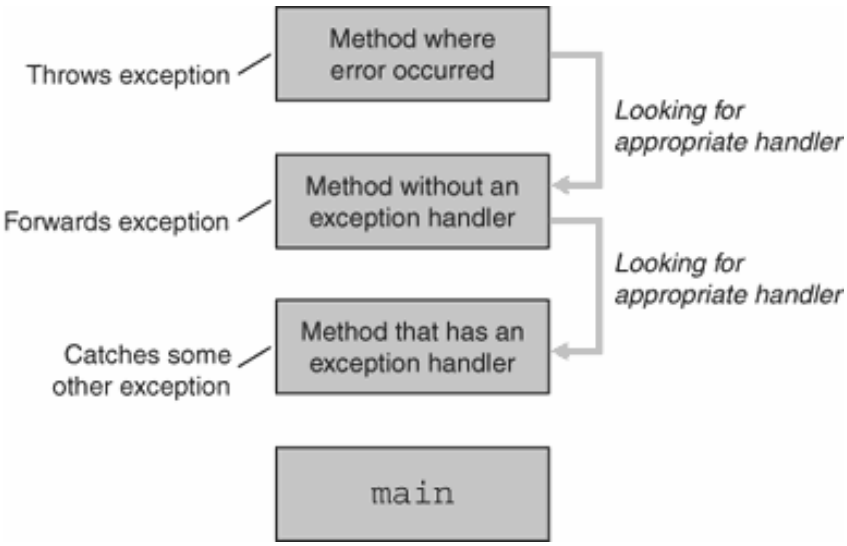
- Some confusion about exceptions
  - Will address it in class
- Slides
- Pace
- Xserver setup
  - Demo in class

Exceptions

- Tool to handle error during program run
  - Exception == exceptional event
  - Idea: when an error occurs, a method can create an Object representing the error and hand it to the run time system
  - The runtime system now tries to find someone to handle the particular error, it uses the call stack to find a handler

Exception handlers

- Are defined by your catch expression
- If a specific method doesn't know how to handle the specific exception, it forwards it up the stack
- Remember: can have multiple catch blocks one after other
  - Exceptions have a hierarchy, they will be evaluated from highest to lowest, so the catch blocks must be in reverse order.



The birth of an exception

- You might use a method which might throw an exception
- You might create a method which creates and exception
- Your code might trigger an exception

InvalidAccountException

```
public class InvalidAccountException extends Exception {


   public InvalidAccountException (String message)
   {
       super(message);
   }

}
```

Your method

```
public boolean checkBalance(int account) throws
   InvalidAccountException{

if(account==null || account < 1){
 throw new InvalidAccountException(“Bad Account
   Number”);
}

... ...
}
```

Chaining Exceptions

```
try {
...
} catch (IOException e) {
   throw new SampleException("Other IOException", e);
}
```

Point

- Can deal with the problem
  - Ask user for help
  - Figure out what should be done
  - Log the error
  - Print a trace to debug
  - Die (ARGHHHHHH!)

Tips

- In a general sense try, catch blocks impose some overhead to the resulting code
- Although can enclose all your code in some try, catch block its not a good idea
- Need to decide at what point, which errors can occur, and what the appropriate response will be

Ahead

- Object Oriented Design

Program Design

- Analysis
- Design
- Implementation

Analysis Phase

- Functional Specification
  - Completely defines tasks to be solved
  - Free from internal contradictions
  - Readable both by domain experts and software developers
  - Reviewable by diverse interested parties
  - Testable against reality

Design Phase

- Goals
  - Identify classes
  - Identify behavior of classes
  - Identify relationships among classes
- Artifacts
  - Textual description of classes and key methods
  - Diagrams of class relationships
  - Diagrams of important usage scenarios
  - State diagrams for objects with rich state

Implementation Phase

- Implement and test classes
- Combine classes into program
- Avoid "big bang" integration
- Prototypes can be very useful

- Object: Three characteristic concepts
  - State
    - Information held by the object
  - Behavior
    - Set of operations supported
  - Identity
    - Unique property setting one object apart from another
- Class: Collection of similar objects

Problem 1:

- Design a voicemail system for use in your typical cellphone.
- How would the requirements look like?
- What would be a typical session?
- What modules are involved?

Identifying Classes in design

- Rule of thumb: Look for nouns in problem description

- Mailbox
- Message
- User
- Passcode
- Extension
- Menu

When defining classes

- Focus on concepts, not implementation

- ????? stores messages
  - Lets say a messageQueue
- Don't worry yet how the queue is implemented

Categories

- Tangible Things
- Agents
- Events and Transactions
- Users and Roles
- Systems
- System interfaces and devices
- Foundational Classes

Identifying Responsibilities

- Rule of thumb: Look for verbs in problem description

- Behavior of MessageQueue:

- Add message to tail
- Remove message from head
- Test whether queue is empty

OO Design

- OO Principle: Every operation is the responsibility of a single class
- Example:
  - Add message to mailbox
- Who is responsible:
  - Message or Mailbox?

Relationship

- Dependency ("uses")
- Aggregation ("has")
- Inheritance ("is")

Dependancy

- C depends on D: Method of C manipulates objects of D

Example: Mailbox depends on Message

- If C doesn't use D, then C can be developed without knowing about D

Independent operations

- Minimize dependency:
  - reduce having to relay on anything set in stone

- Example: Replace

void print() // prints to System.out

- with

String getText() // can print anywhere

- Removes dependence on System, PrintStream

Aggregation

- Object of a class contains objects of another class
- Example: MessageQueue aggregates Messages
- Example: Mailbox aggregates MessageQueue
- Implemented through instance fields

Relationships

- 1 : 1 or 1 : 0...1 relationship:

```
public class Mailbox
{
   . . .
   private Greeting myGreeting;
}
```

- 1 : n relationship:

```
public class MessageQueue
{
   . . .
   private ArrayList<Message> elements;
}
```

Inheritance

- More general class = superclass
- More specialized class = subclass
- Subclass supports all method interfaces of superclass (but implementations may differ)
- Subclass may have added methods, added state
- Subclass inherits from superclass
- Example:
  - ForwardedMessage inherits from Message
  - Greeting does not inherit from Message (Can't store greetings in mailbox)

Use Cases

- Analysis technique
- Each use case focuses on a specific scenario
- Use case = sequence of actions
- Action = interaction between actor and computer system
- Each action yields a result
- Each result has a value to one of the actors
- Use variations for exceptional situations

Example case

- Leave a Message

1. Caller dials main number of voice mail 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

5. Caller speaks message
6. Caller hangs up
7. System places message in mailbox

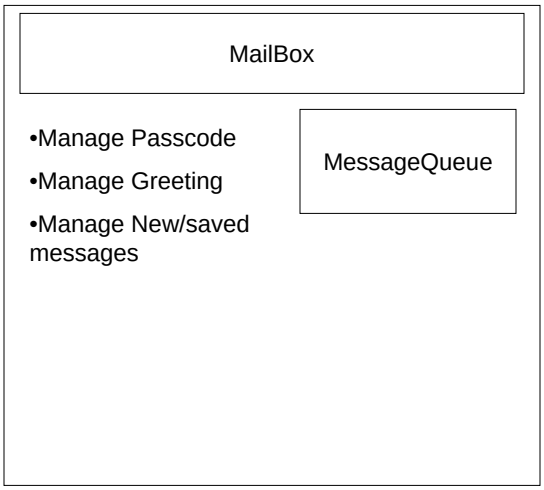
Variations

- user enters invalid extension number
  - What do you do?
  - Who does it?
- What if user hangs up instead of using message?
- How many attempts at password?

CRC Cards

- CRC = Classes, Responsibilities, Collaborators
- Developed by Beck and Cunningham
- Use an index card for each class
- Class name on top of card
- Responsibilities on left
- Collaborators on right

CRC



- Responsibilities should be high level
- 1 - 3 responsibilities per card
- Collaborators are for the class, not for each responsibility

Example

- Use case: "Leave a message"
- Caller connects to voice mail system
- Caller dials extension number
- "Someone" must locate mailbox
- Neither Mailbox nor Message can do this
- New class: MailSystem
- Responsibility: manage mailboxes

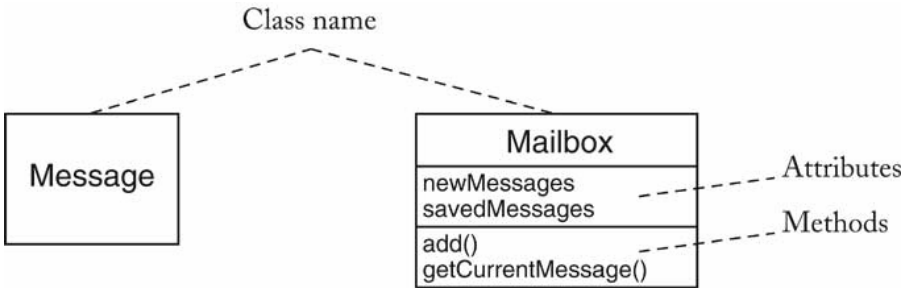
UML

- UML = Unified Modeling Language
- Unifies notations developed by Booch, Rumbaugh, Jacobson
- Many diagram types
- We'll use three types:
  - Class Diagrams
  - Sequence Diagrams
  - State Diagrams

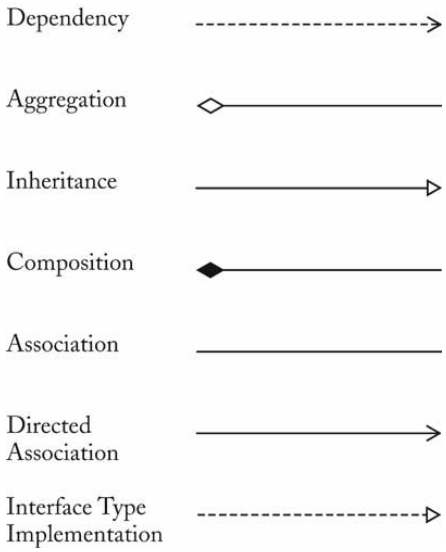
Class Diagrams

- Rectangle with class name
- Optional compartments
  - Attributes
  - Methods
- Include only key attributes and methods

Class Diagram

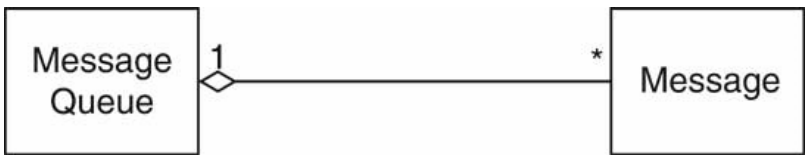


UML Relationships



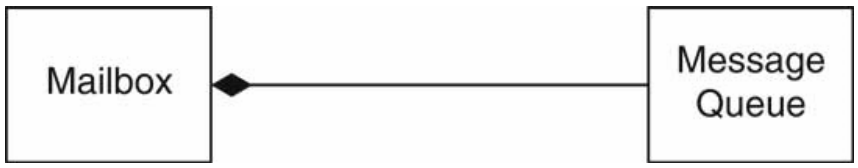
Multiplicities

- any number (0 or more): *
- one or more: 1..*
- zero or one: 0..1
- exactly one: 1



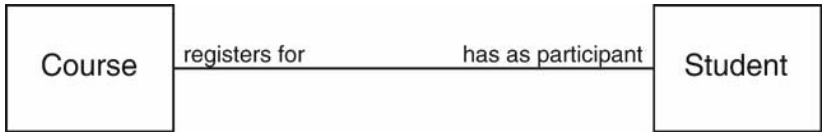
Composition

- Special form of aggregation
- Contained objects don't exist outside container
- Example: message queues permanently contained in mail box



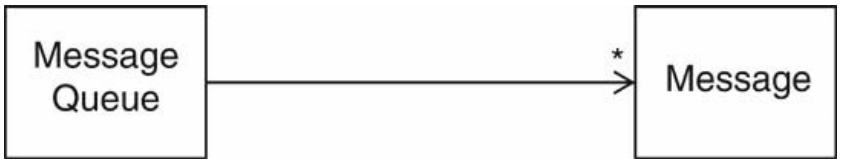
Association

- Some designers don't like aggregation
- More general association relationship
- Association can have roles



Association II

- Some associations are bidirectional
- Can navigate from either class to the other
- Example: Course has set of students, student has set of courses
- Some associations are directed
- Navigation is unidirectional
- Example: Message doesn't know about message queue containing it



Interface Types

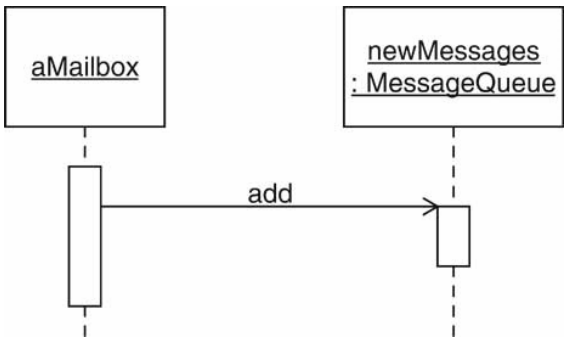
- Interface type describes a set of methods
- No implementation, no state
- Class implements interface if it implements its methods
- In UML, use stereotype «interface»

Tip

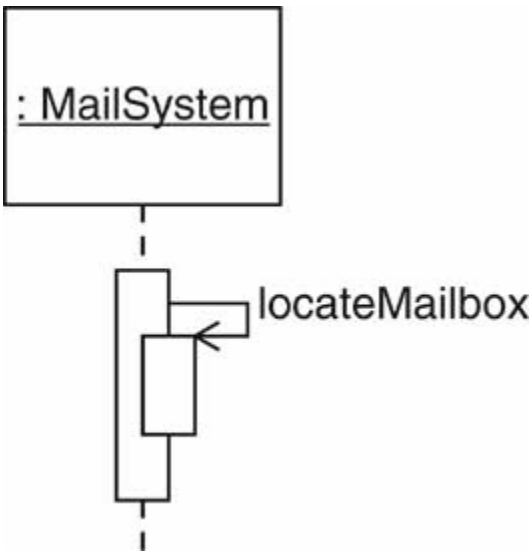
- Use UML to inform, not to impress
- Don't draw a single monster diagram
- Each diagram must have a specific purpose
- Omit inessential details

Sequence Diagrams

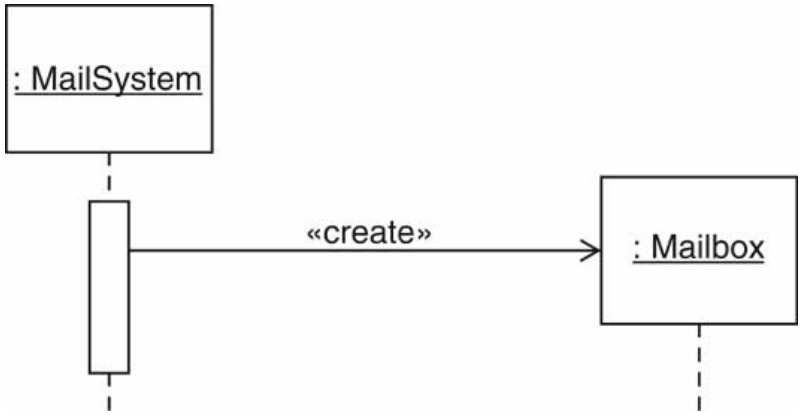
- Each diagram shows dynamics of scenario
- Object diagram: class name underlined



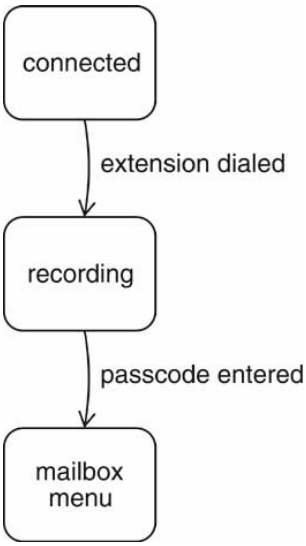
Self call



Object Construction



State Diagram



Design Docs

- Recommendation: Use Javadoc comments
- Leave methods blank

```
/**
   Adds a message to the end of the new messages.
   @param aMessage a message
*/
public void addMessage(Message aMessage)
{
}
```

- Don't compile file, just run Javadoc
- Makes a good starting point for code later

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

Next time

- Read
- Make sure you are making headway in the homework
- Download UML designer and try to play with it.