Object-oriented Software Design and Development
CCP114N

Week 3:

The Object - Oriented Software Design Process

Topics

- Software Analysis-Design-Implementation
- What are Objects and Classes
  - How to identify them
- Use Cases
- CRC Cards
- UML Diagrams
- Using **javadoc** for Design Documentation

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 behaviour of classes
    - Identify relationships among classes
  - Artefacts
    - 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 and Class Concepts
Identifying Classes

- **Object**: three characteristic concepts
  - State
  - Behaviour
  - Identity
- **Class**: Collection of similar objects
- **Identifying Classes**
  - Rule of thumb: Look for *nouns* in problem description e.g. Mailbox, Message, User, Passcode, Extension, Menu
  - Focus on *concepts*, not implementation, *something that plays a role that has some responsibility*
    - MessageQueue stores messages
    - Don't worry yet how the queue is implemented
  - Categories of Classes
    - 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
- E.g. Behavior of MessageQueue:
  - Add message to tail
  - Remove message from head
  - Test whether queue is empty
- Responsibilities

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

Class Relationships

Dependency, Aggregation and Inheritance

Dependency Relationship ("uses")

- 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
- **Coupling**
  - Minimize dependency: reduce *coupling*
  - Example: Replace

    void print() // prints to System.out

    with **String getText() // can print anywhere**

    Removes dependence on System, PrintStream

Class Relationships

Dependency, **Aggregation** and Inheritance

Aggregation (“has”)

- Object of a class contains objects of another class
- Example: MessageQueue aggregates Messages
- Example: Mailbox aggregates MessageQueue
- Implemented through instance fields
- **Multiplicities**
  - 1 : 1 or 1 : 0...1 relationship:

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

  - 1 : *n* relationship:

```
public class MessageQueue
   {
      . . .
      private ArrayList elements;
   }
```

Class Relationships

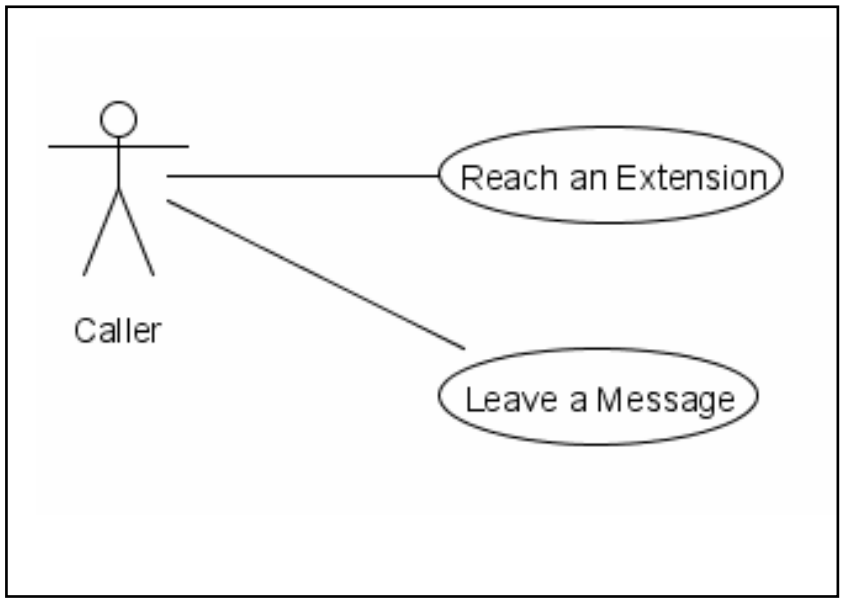
Dependency, Aggregation and **Inheritance**

Inheritance (“is”)

- 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
- Example: 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



Use Cases: Sample Use Case

- **Leave a Message**
  - Caller dials main number of voice mail system
  - System speaks prompt
  - Enter mailbox number followed by #
  - User types extension number
  - System speaks
  - You have reached mailbox xxxx. Please leave a message now
  - Caller speaks message
  - Caller hangs up
  - System places message in mailbox
- **Sample Use Case -- Variations**
  - **Variation #1**

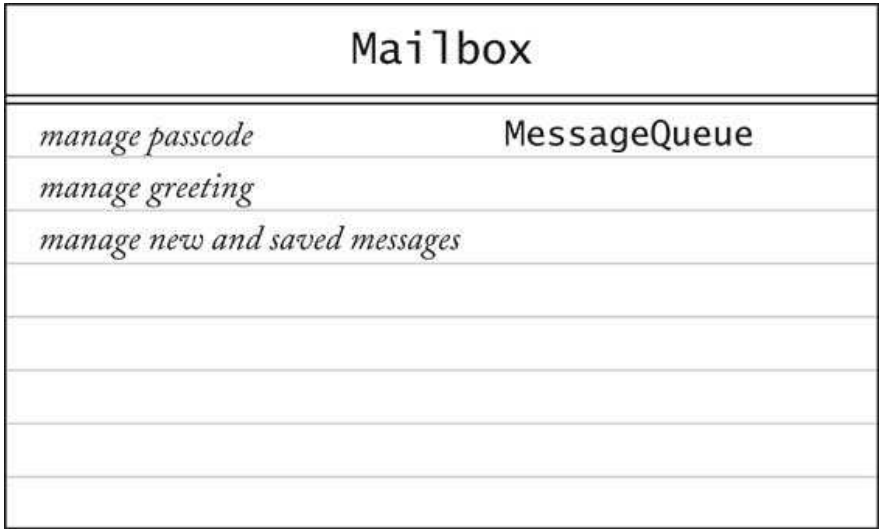
    1.1. In step 3, user enters invalid extension number
    1.2. Voice mail system speaks: You have typed an invalid mailbox number.
    1.3. Continue with step 2.
  - **Variation #2**

    2.1. After step 4, caller hangs up instead of speaking message
    2.3. Voice mail system discards empty message

CRC Cards

CRC = Classes – Responsibilities - Collaborators

- Use an index card for each class (not UML, but useful)
- Class name on top of card
- Responsibilities on left
  - Responsibilities should be *high level*
  - 1 - 3 responsibilities per card
- Collaborators on right
  - Collaborators are for the class, not for each responsibility
- **Walkthroughs**
  - Using Use Cases and CRC cards, e.g.
  - 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 discovered: MailSystem
    - Responsibility: manage mailboxes

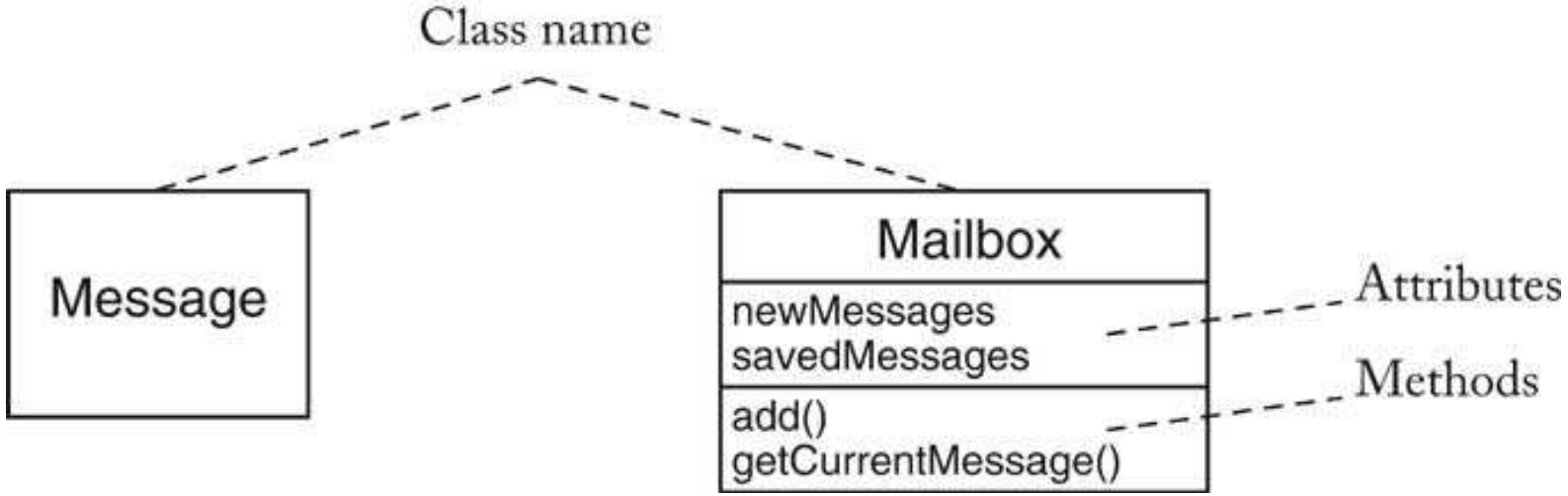


UML – Unified Modeling Language

- Unifies notations developed by the "3 Amigos" Booch, Rumbaugh, Jacobson
- UML books
  - The Unified Modeling Language User Guide, by Grady Booch, James Rumbaugh, Ivar Jacobson; Addison-Wesley
  - The Unified Software Development Process, Booch, G; Rumbaugh, J and Jacobson, I. ; Addison Wesley
- Many diagram types: 9 of them
- We'll use three types:
  - Class Diagrams
  - Sequence Diagrams
  - State Diagrams

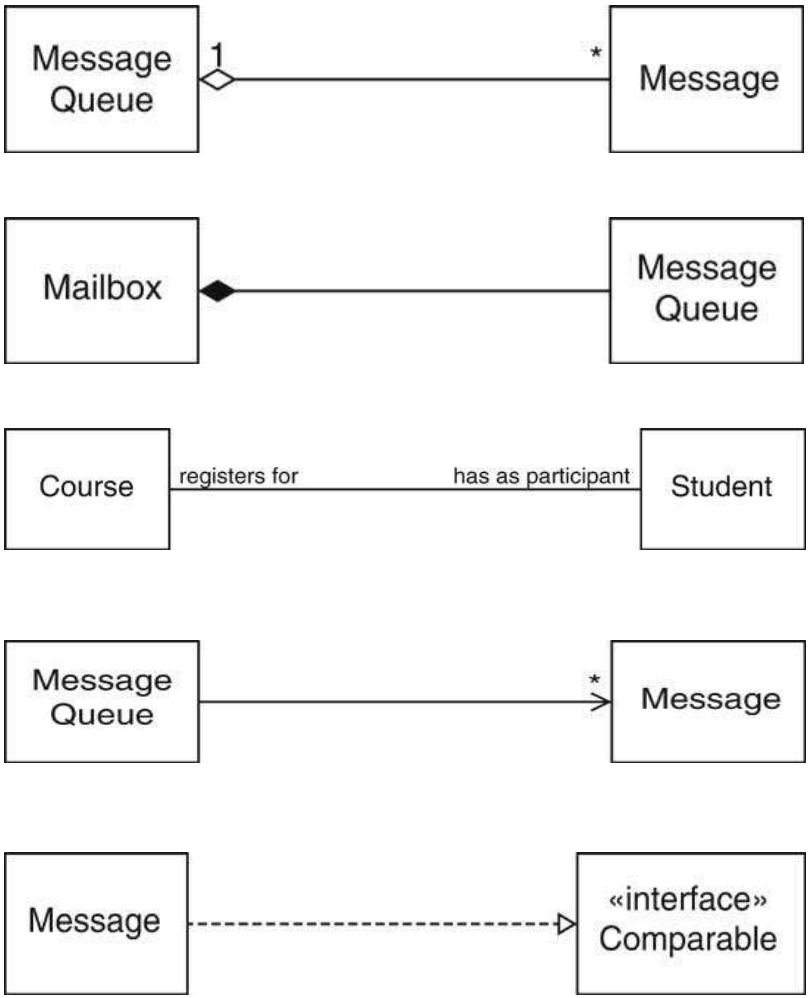
Class Diagrams

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



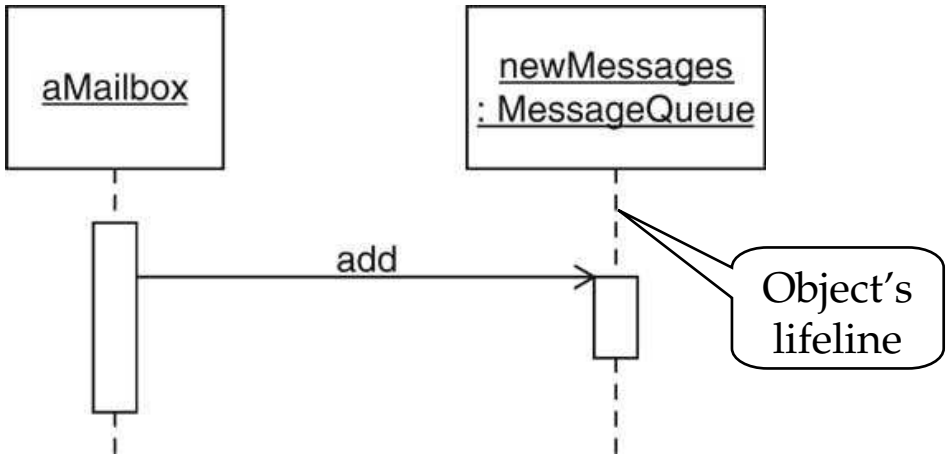
Class Relationships

- **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**
  - More general association relationship
  - Association can have roles
  - 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»

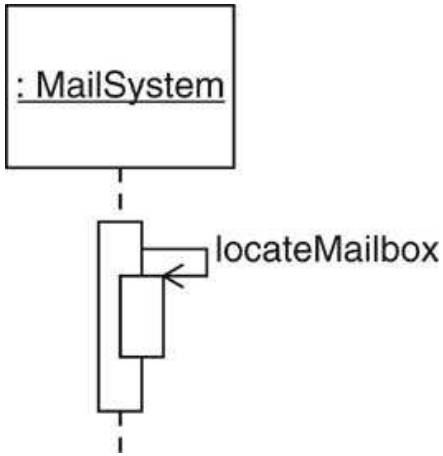


Sequence Diagrams

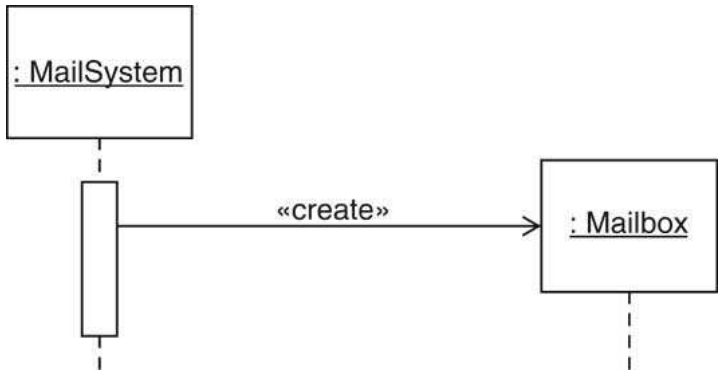
- Each diagram shows dynamics of scenario in terms of its flow control.
- Object diagram: similar to class diagram with class name underlined



- **Self call**



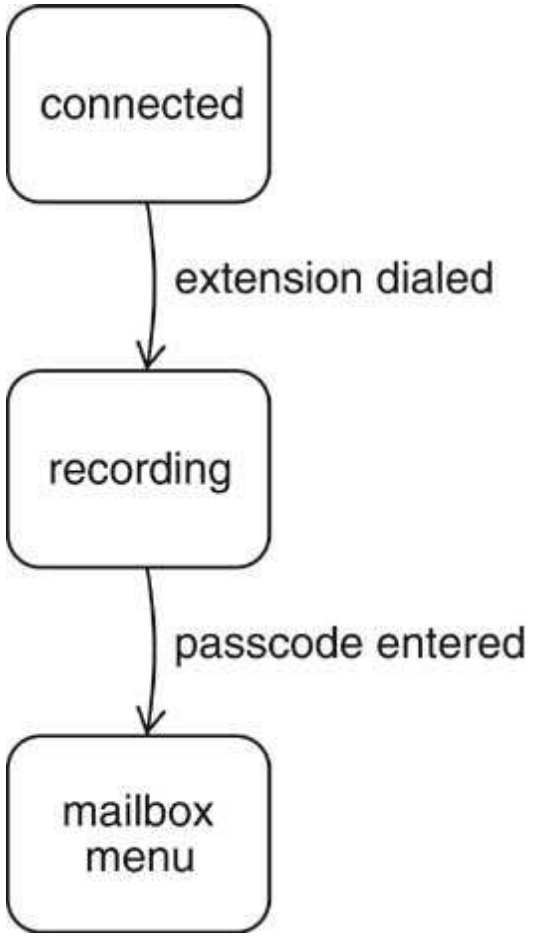
- **Object Construction**



State Diagram

- Use for classes whose objects have interesting states
  - Object behaviour may depend on its (internal) states e.g. a message cannot be recorded if the caller is not connected.
- Represents an object's states (rounded rectangles) and transitions between the states (arrows)
- Example:

  Connection State Diagram



Design Documentation

- Documentation formats may be varied
- Recommendation: May use Javadoc comments as in the template below

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

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

Summary

- Phases of the software system process
- Software system Classes and Objects
- Identifying classes
- CRC cards and use cases
- UML diagrams
- Documenting Class Design