# Chapter One

Introduction to Object Oriented Technologies

# Course Outline

- **Chapter 1: Introduction to Object Oriented Technologies**
- **Chapter 2: Core Object Oriented Technologies with Java**
- **Chapter 3: Design with UML**
- **Chapter 4: Design Principles**
- **Chapter 5: Design Patterns**

# Chapter Outlines

- Five attributes of a complex system
- Object Oriented Technologies (OOT)
  - Object Oriented Design (OOD)
  - Object Oriented Programming (OOP)
- Object Oriented Concepts
- Object Models
- CRC card

# Software Development Life Cycle

- SDLC is a structured sequence of stages in software engineering to develop the software product
- It involves the following steps
  - Define User Requirements
  - Feasibility Study (Proof of Concept, POC)
  - System Analysis
  - Software Design
  - Programming
  - User Acceptance Testing, UAT
  - Operations & Maintenance

# The complexity of the problem domain

- Software often involves elements of complexity, in which we find a plenty of competing, perhaps even contradictory, requirements
- The “communication gap” that exists between the users of a system and its developers
- Users and developers have different perspectives on the nature of the problem

# The difficulty of managing the development process

- Users generally find it very hard to give precise expression to their needs in a form that developers can understand
- The user requirements of a software system often change during its development
- Considering the nature of complexity, there are five attributes common to all complex systems
  - Hierarchic Structure, Relative Primitives, Separation of Concerns, Common Patterns, Stable Intermediated Forms

# Five Attributes of a Complex System

Common five attributes:

1. Hierarchic Structure
   - All systems have subsystems and all systems are parts of larger systems
2. Relative Primitives
   - The choice of what components in a system are primitive is relatively arbitrary and is largely up to the discretion of the observer of the system

# Five Attributes of a Complex System (cont.)

3. Separation of Concerns
   - Provides a clear *separation of concerns* among the various parts of a system, making it possible to study each part in relative isolation
4. Common Patterns
   - Complex systems have common patterns. These patterns may involve the reuse of small components
5. Stable Intermediate Forms
   - We can never craft the primitive objects correctly at the first time, and complex systems tend to evolve over time

# Problem Solving

- There are different ways to write programs for solving the same problem. Which approach will you use?
  - Quick-and-dirty code (it complies, it works)
  - Copy-and-paste
  - Others
- Have you ever looked back your developed programs?
- Do you think your codes can be reused by other people?

# When similar problems arise

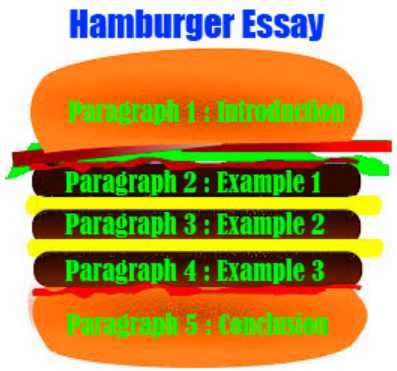
- If you need to handle a similar problem that you have already solved before, what will you do?
  - Rewrite a new program from scratch
  - Copy the old one and modify the outstanding issues
- Is it a better way to reuse the developed programs for future projects?
- Object Oriented Technologies is the answer
  - The techniques to write codes ELEGANT
  - The ideas to design codes for REUSE
  - The concept to GENERALIZE the solutions for solving different problems

# Is it really necessary?

- Programmers can write limited codes per day
- Debugging programs is time consuming
- A good design can
  - save time on writing duplicated codes
  - break down the problem into smaller pieces
  - avoid bugs easier and minimize potential risks
  - separate functional modules for individual team member to reduce the workload
  - make enhancement easily

# Writing better software

- Developing a software is like writing an essay
- What will you do to write an essay?
  - Find templates that fit the requirements
  - Hamburger format is a good choice
  - Fill each paragraph with a point
- OOT help people to design and develop a better software with the following properties
  - usability, completeness, robustness, efficiency, scalability, readability, reusability, simplicity, maintainability, and extensibility



# Good Software Properties

- Usability
  - Is it easy for clients to use?
- Completeness
  - Does it satisfy all client's needs?
- Robustness
  - Will it deal with unusual situations gracefully and avoid crashing?
- Scalability
  - Will it still perform correctly and efficiently when the problems grow in size by several orders of magnitude?

# Good Software Properties (cont.)

- Efficiency
  - Will it perform the necessary computations in a reasonable amount of time and using a reasonable amount of memory and other resources?
- Readability
  - Is it easy for another programmer to read and understand the design and code?
- Reusability
  - Can it be reused in another completely different setting?

# Good Software Properties (cont.)

- Simplicity
  - Is the design and/or implementation unnecessarily complex?
- Maintainability
  - Can defects be found and fixed easily without adding new defects?
- Extensibility
  - Can it easily be enhanced or restricted by adding new features or removing old features without breaking code?

# Novice Programming Style

- People often write a single lengthy program to do everything for solving a problem
- Problems of all-in-one approach
  - It contains logic, data, user interface together
  - It is hard to read, debug, expend, etc.
  - It is not reusable
  - We called it: “The spaghetti code”
- Procedural Programming breaks problems into steps and each step may contain smaller steps
- There are many small steps with similar purposes and functions. They are hardly to be reused

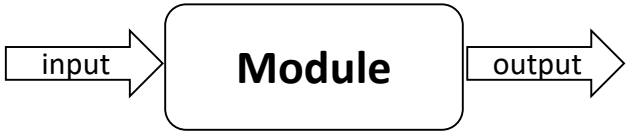
# Procedural Programming Approach

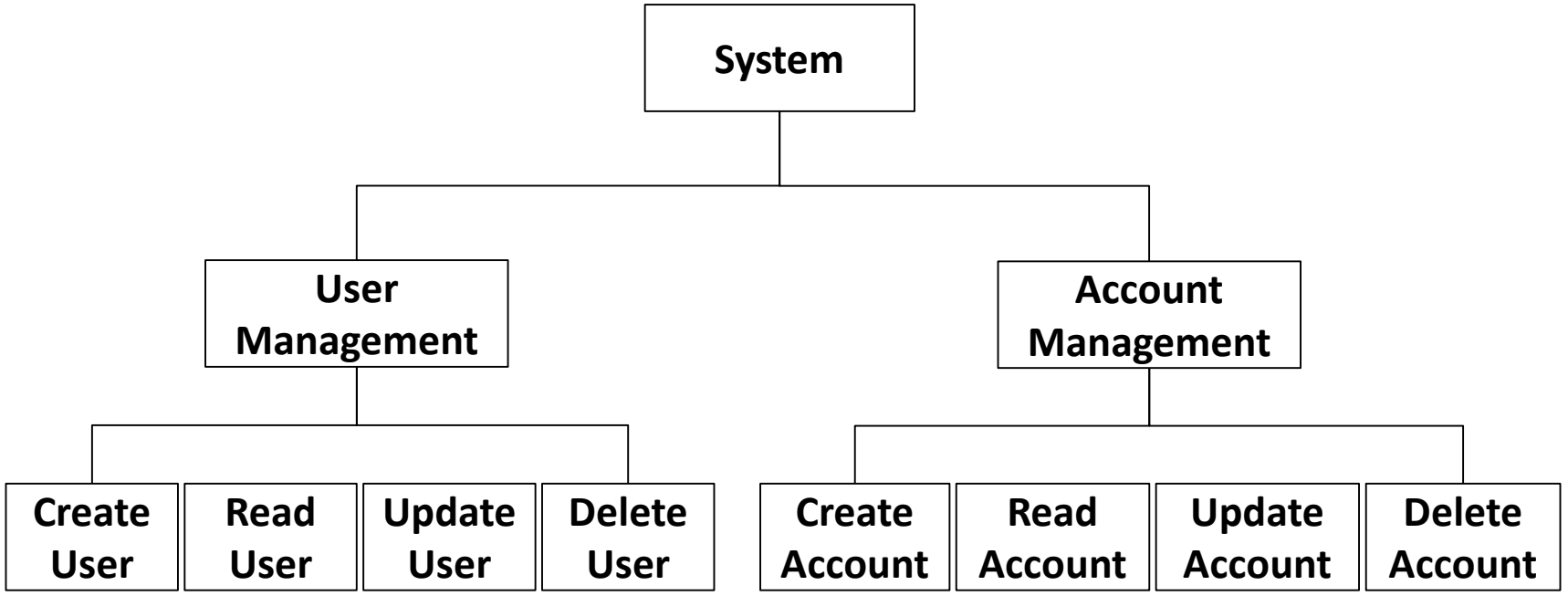
- Procedural Programming Style
  - The concept of calling the procedures (routines, subroutines, functions, methods, etc.)

```
01.  public class ProcedureProgram {
02.    public ProcedureProgram() {
03.      step1();
04.      step2();
05.      step3();
06.    }
07.    private void step1() { … }
08.    private void step2() { … }
09.    private void step3() { … }
10.    public static void main(String args[]) {
11.      new ProcedureProgram();
12.    }
13.  }
```

# Function Oriented Design

- FOD identifies system functions and transform them with input and output



# FOD Characteristics

- FOD approach defines module base on functions
  - User Management, Account Management, etc.
- Each module is then divided into sub modules
  - Create user, query user, update account, delete account, etc.
- This top-down approach allows sub modules to be reused by other modules
  - Account Management needs to check user information, etc.
- However, when module A changes, sub module B may not be reused anymore

# Object Oriented Design

- The approach of defining everything in terms of object
  - User Object, Account Object, etc.
- Each object contains encapsulated data and procedures grouped together
  - User Name, Password, upgradeToVIP(), freezeAccount(), etc.
- Objects are assembled together and interacted with others, and eventually become a complete system
  - Objects are the building block
- This bottom-up approach help to let objects become more independent and flexible

# OO Programming Approach

- It is similar as producing puzzle
  - Designer (team leader) designs a blueprint for the puzzle (system) first
  - Designer defines each piece of puzzle as an object
  - Programmers build objects
- Finally, players (development team) assemble them together
  - Play with objects

# OOP by Definition

"Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some classes, and whose classes are all members of a hierarchy of classes united via inheritance relationships."

*Object-Oriented Analysis and Design with Application, 3rd Edition*. Addison Wesley

# What exactly it is?

- OO uses objects as its fundamental building blocks
- Each object is an instance of some classes
- Classes may be related to one another via inheritance relationships
- OO programming languages support classes, objects, messages, inheritance, polymorphism, etc.
- It is not OO if the ***object model*** doesn't have the four major elements
  - Abstraction, Encapsulation, Modularity, Hierarchy

# Object Model: Abstraction

- An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer
- Abstraction focuses on the outside view of an object and so serves to separate an object's essential behavior from its implementation
- object B (subclass) "is a" object A (superclass)
  - Domestic cats are Felidae
  - Lions and tigers are Felidae

# Object Model: Encapsulation

- For abstraction to work, implementations must be encapsulated
- Encapsulation hides the abstraction details from the end users
- Abstraction focuses on the observable behavior of an object, where as encapsulation focuses on the implementation that gives rise to this behavior

# Object Model: Modulation

- Modulation divides a program into modules which can be compiled separately, but which have connections with other modules
- Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules
- ***Cohesive***: by grouping logically related abstractions
- ***Loosely Coupled***: by minimizing the dependencies among modules

# Object Models: Hierarchy

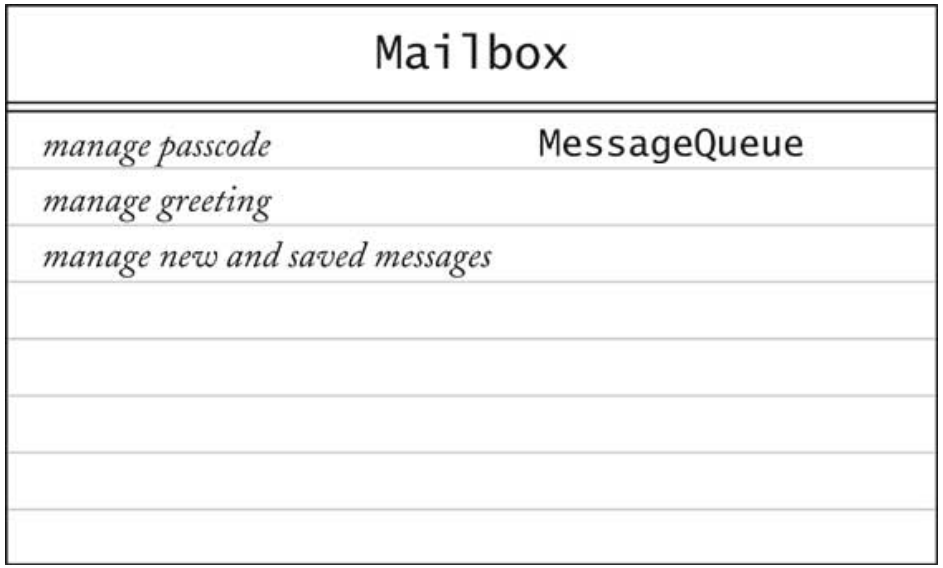
- Hierarchy is a ranking or ordering of abstractions
- The two most important hierarchies in a complex system are its class structure (the “is a” hierarchy) and its object structure (the “part of” hierarchy)
- The “is a” hierarchies denote generalization or specialization relationships, “part of” hierarchies describe ***aggregation*** relationships
- A garden consists of a collection of plants with a growing plan. Plants are “part of” the garden, and the growing plan is “part of” the garden

# Benefits of using Object Model

- It helps us to exploit the expressive power of object-based and object oriented programming languages
- It encourages the reuse not only of software but of entire designs, leading to the creation of reusable application frameworks
- It produces systems that are built on stable intermediate forms, which are more resilient to change
- It appeals to the workings of human cognition

# Design software with CRC Card

- Classes, Responsibilities, Collaboration (CRC)
  - An index card for a class
  - Help to design things in a class
- Class name (top)
- Responsibilities (left)
- Collaborators (right)
  - Other classes that need to collaborate with so that it can fulfill its responsibilities



# CRC Card

- Small
  - Discourage you from piling too much responsibility into a single class
- Low-tech
  - So they can be used by groups of designers gathered around a table
- Rugged than sheets of paper
  - Can be handed around and rearranged during brainstorming sessions
- Responsibilities should be *high level*
  - DON’T write individual methods
- Collaborators are for class but not for responsibility
  - Simply list collaborators as you discover them, without regard for ordering

# Summary

- The maturation of software engineering has led to the development of object oriented analysis, design and programming methods
- The object oriented technologies help us to develop large and complex systems more easily
- The object model mainly consists of abstraction, encapsulation, modularity, and hierarchy