Object Oriented Analysis & Design (OOAD) Overview

Lecture 11

Based on slides from Ian Sommerville, University of Lancaster, United Kingdom, University of St Andrews, Scotland

And on slides Aries Muslim's site (based on slides from Kasseler?)

Object-Oriented Development

- Object-oriented analysis, design and programming are related but distinct
  - OOA is concerned with developing an object model of the application domain
  - OOD is concerned with developing an object-oriented system model to implement requirements
  - OOP is concerned with realising an OOD using an OO programming language such as Java or C++

OOAD

- **Focuses on objects** where system is broken down in terms of the objects that exist within it.

- Functions (**behaviour**) and data (**state**) relating to a single object are self-contained or encapsulated in one place.

Advantages of OOD

- Easier maintenance. Objects may be understood as stand-alone entities
- Objects are appropriate reusable components
- For some systems, there may be an obvious mapping from real world entities to system objects

OO Design vs. OO Programming

- Object-Oriented Design
  - a method for decomposing software architectures
  - based on the objects every system or subsystem manipulates
  - relatively independent of the programming language used

- Object-Oriented Programming
  - construction of software systems as
    - Structured collection of Abstract Data Types (ADT)
    - Inheritance
    - Polymorphism
  - concerned with programming languages and implementation issues

Objects and object classes

- Objects are entities in a software system that represent instances of real-world and system entities
- Object classes are templates for objects. They may be used to create objects
- Object classes may inherit attributes and services from other object classes

Objects

- Object is an abstraction of something in a problem domain, reflecting the capabilities of the system to keep information about it, interact with it, or both.
- Objects are entities in a software system that represent instances of real-world and system entities

Objects

Object	Identity	Behaviors	State
An employee	"Mr. John"	Join(), Retire()	Joined, Retired.
A book	"Book with title Object Oriented Analysis Design"	AddExemplar,	Rent, available, reserved
A sale	"Sale no 0015, 15/12/98"	SendInvoice(), Cancel().	Invoiced, cancelled.

Object Class

- Class is a description of a set of objects that share the same attributes, operations, methods, relationship and semantics.
- Object classes are templates for objects. They may be used to create objects.
- An object represents a particular instance of a class.

Term of objects

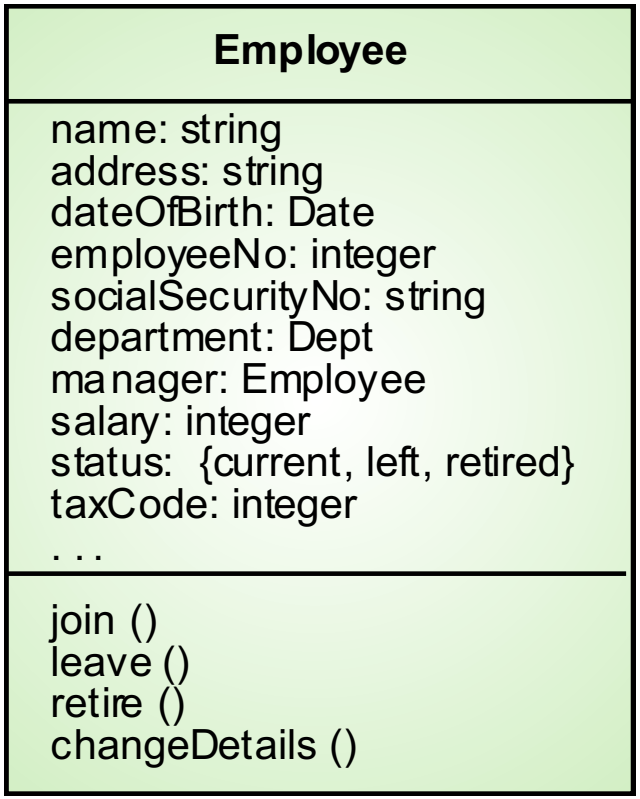
- Attributes: data items that define object.
- Operations: functions in a class that combine to form the behavior of class.
- Methods: the actual implementation of a procedure (the body of code that is executed in response to a request from other objects in the system).

The Unified Modeling Language

- Several different notations for describing object-oriented designs were proposed in the 1980s and 1990s
- The Unified Modeling Language is an integration of these notations
- It describes notations for a number of different models that may be produced during OO analysis and design
- It is now a *de facto* standard for OO modelling

Employee object & class

Class



Object

Employee16

name: John
address: M Street No.23
dateOfBirth: 02/10/65
employeeNo: 324
socialecurityNo:E342545
department: Sale
manager: Employee1
salary: 2340
stauts:current
taxCode: 3432
….

Eployee16.join(02/05/1997)
Eployee16.retire(03/08/2005)
Eployee16.changeDetail("X Street No. 12")

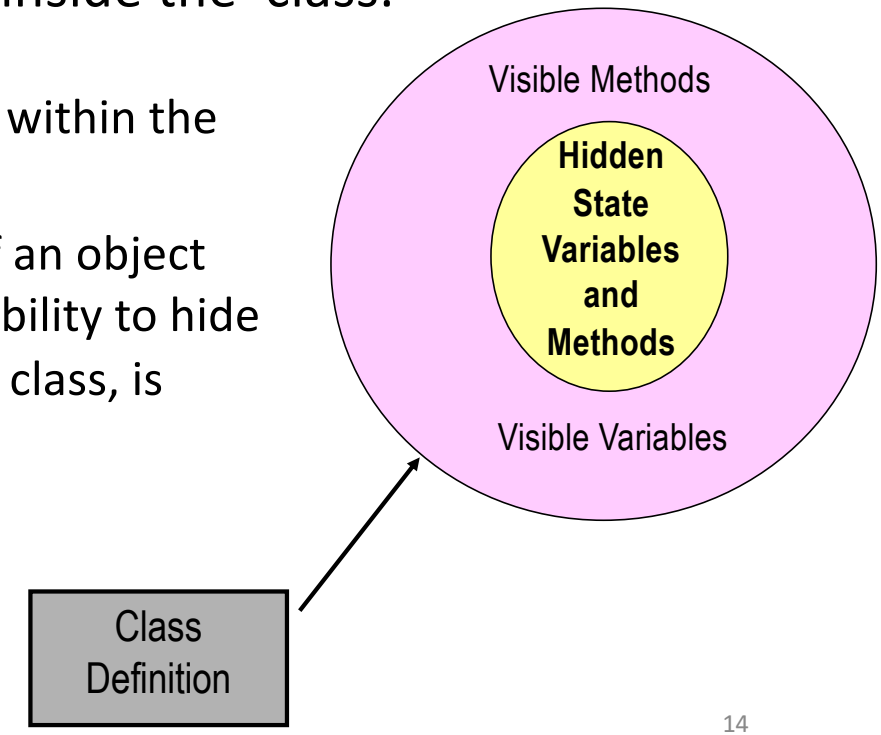
Encapsulation and Data Hiding

- Packaging related data and operations together is called encapsulation.
- Data hiding: hides the internal data from external by methods (interface).
- Important in most design paradigms (not just OOAD)

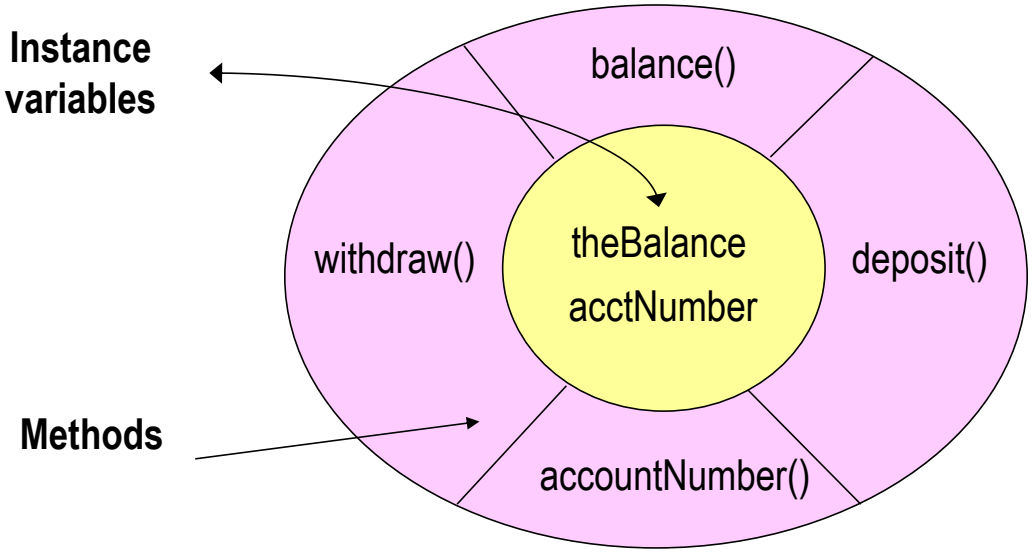
Encapsulation

When classes are defined, programmers can specify that certain methods or state variables remain hidden inside the class.

- These variables and methods are accessible from within the class, but not accessible outside it.
- The combination of collecting all the attributes of an object into a single class definition, combined with the ability to hide some definitions and type information within the class, is known as encapsulation.



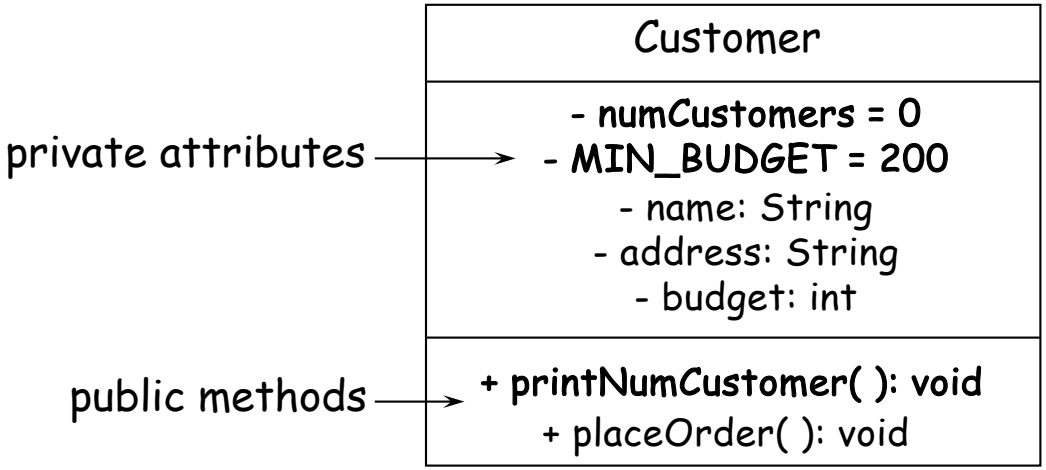
Graphical Model of an Object



State variables make up the nucleus of the object. Methods surround and hide (encapsulate) the state variables from the rest of the program.

Encapsulation

- ***private*** **attributes and methods are encapsulated within the class, they cannot be seen by clients of the class**
- ***public*** **methods define the interface that the class provides to its clients**



Customer class

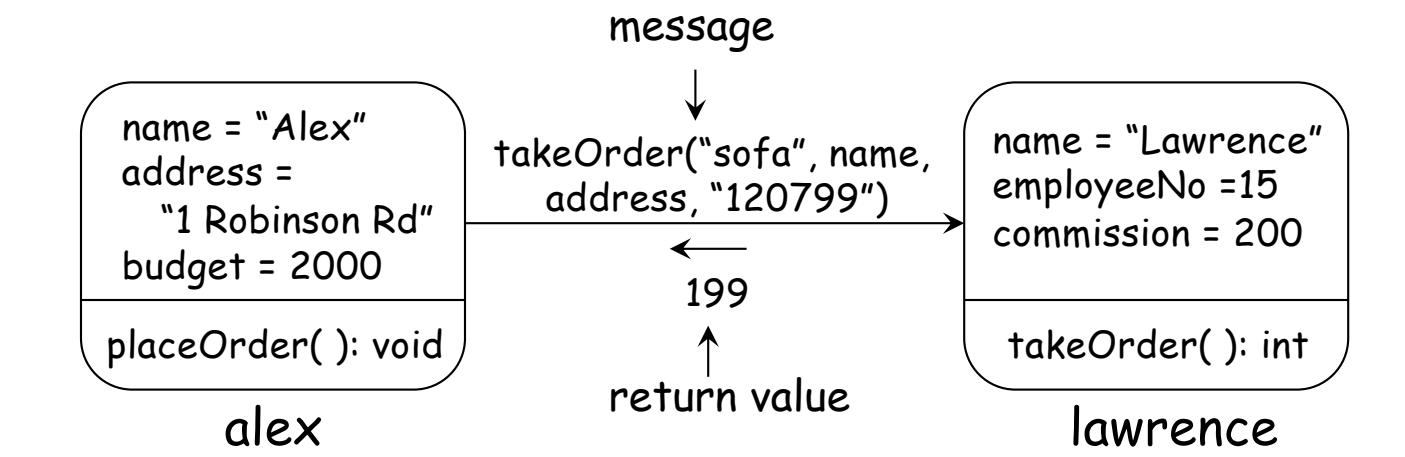
Object communication

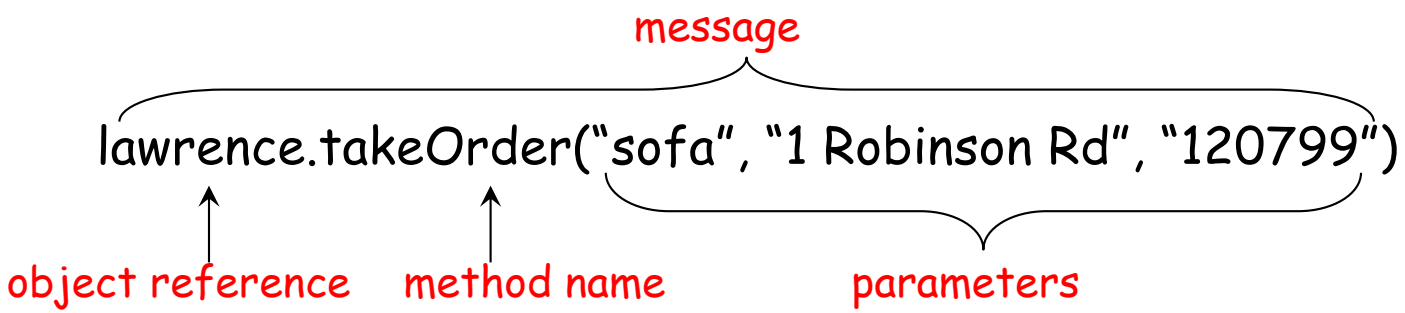
- Conceptually, objects communicate by message passing.
- Messages
  - The name of the service requested by the calling object.
  - Copies of the information required to execute the service and the name of a holder for the result of the service.
- In practice, messages are often implemented by procedure calls
  - Name = procedure name.
  - Information = parameter list.

Object communication

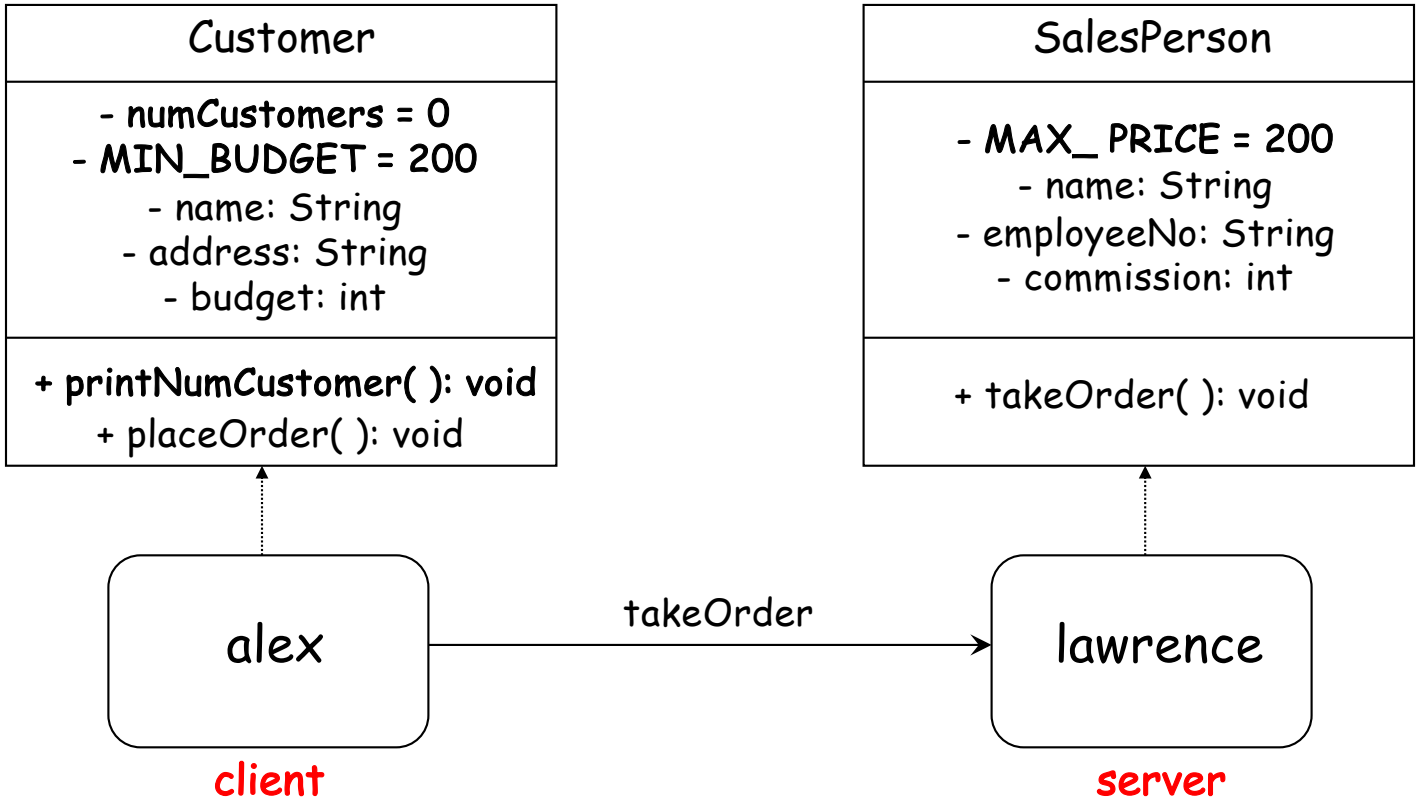
- Objects communicate with each other by sending *messages*
  - a message is a method call from a message-sending object to a message-receiving object
  - a message consists of
    - an *object reference* which indicates the message receiver
    - a *method name* (corresponding to a method of the receiver), and
    - *parameters* (corresponding to the arguments of the calling method)
  - a message-receiving object is a *server* to a message-sending object, and the message-sending object is a *client* of the server

Message Passing





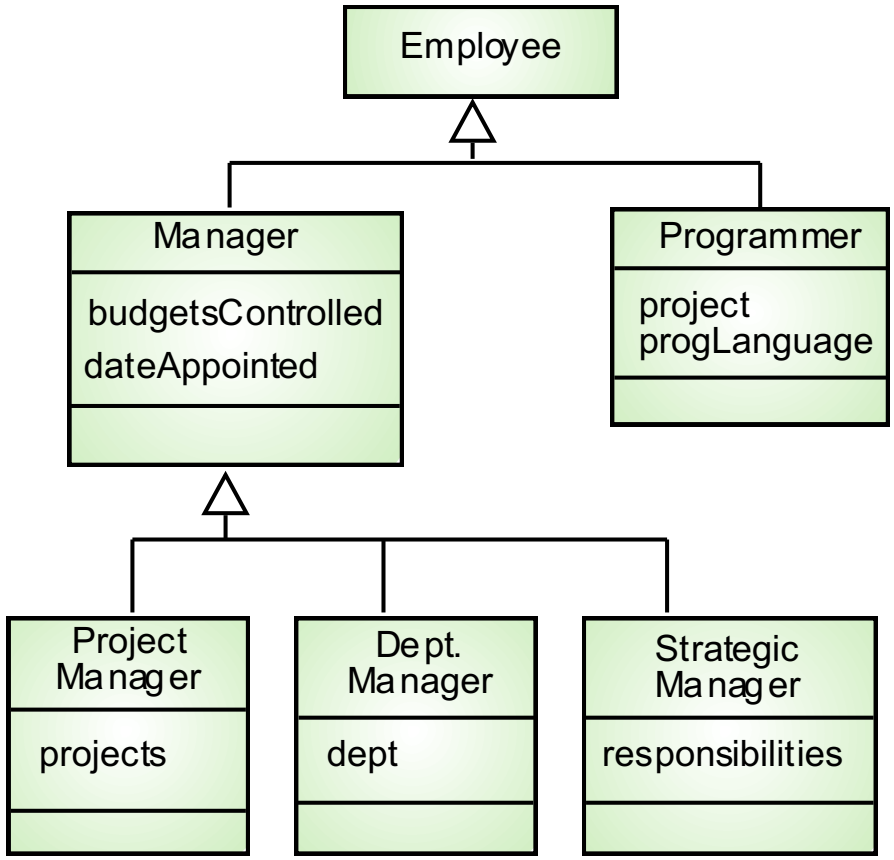
Message Passing



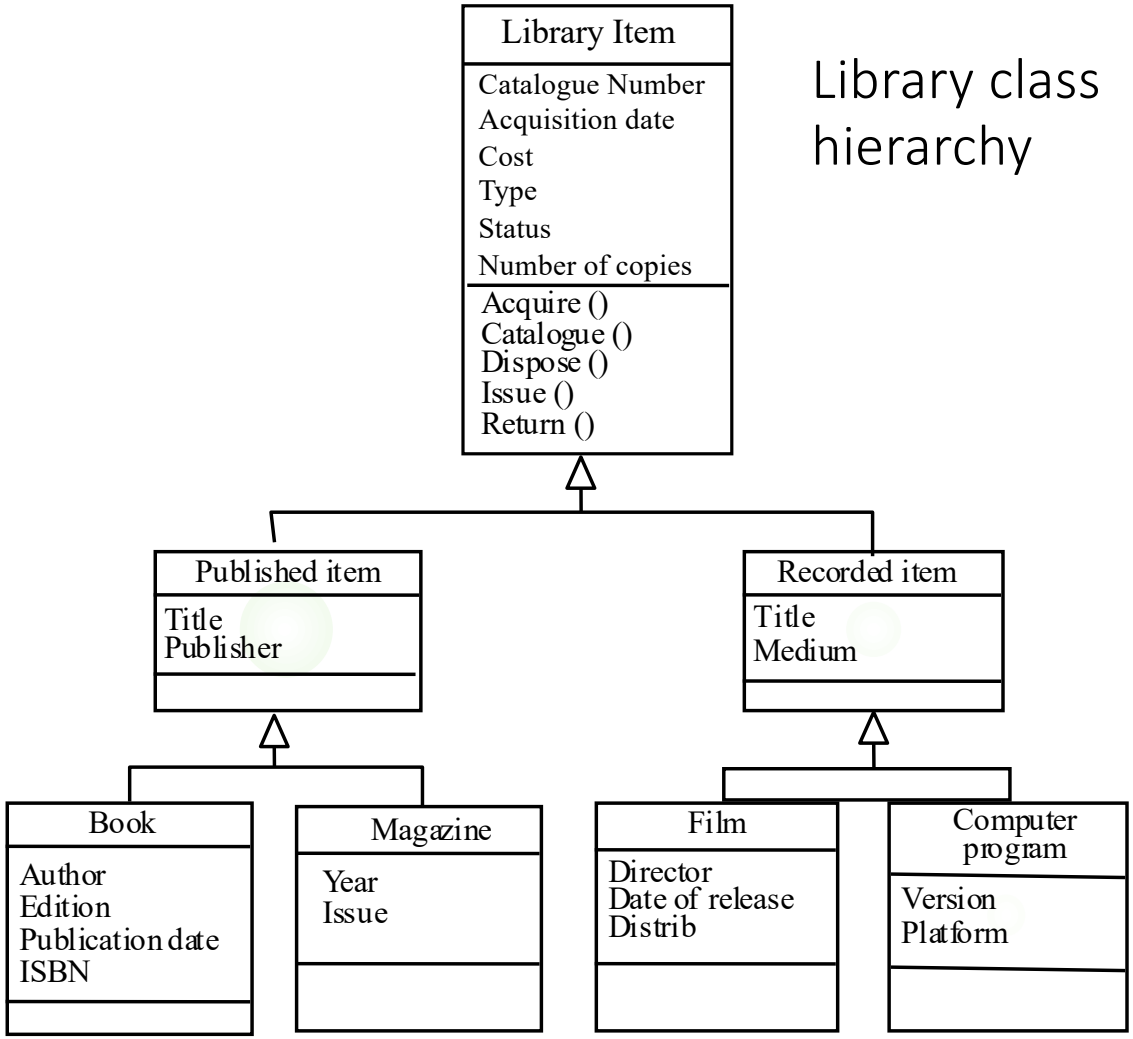
Generalisation and inheritance

- Objects are members of classes that define attribute types and operations
- Classes may be arranged in a class hierarchy where one class (a super-class) is a generalisation of one or more other classes (sub-classes)
- A sub-class inherits the attributes and operations from its super class and may add new methods or attributes of its own
- Generalisation in UML is implemented as inheritance in OO programming languages

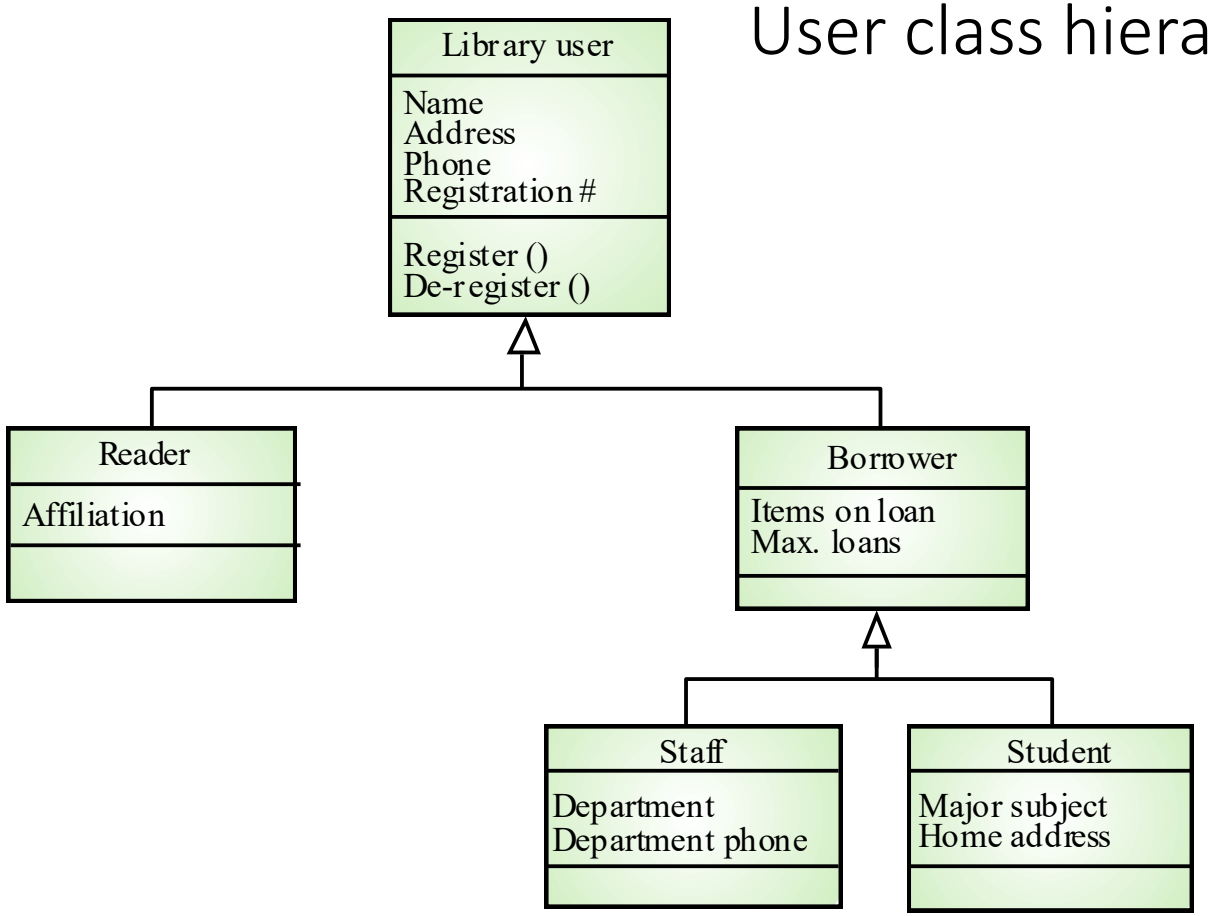
A generalisation hierarchy



Library class hierarchy



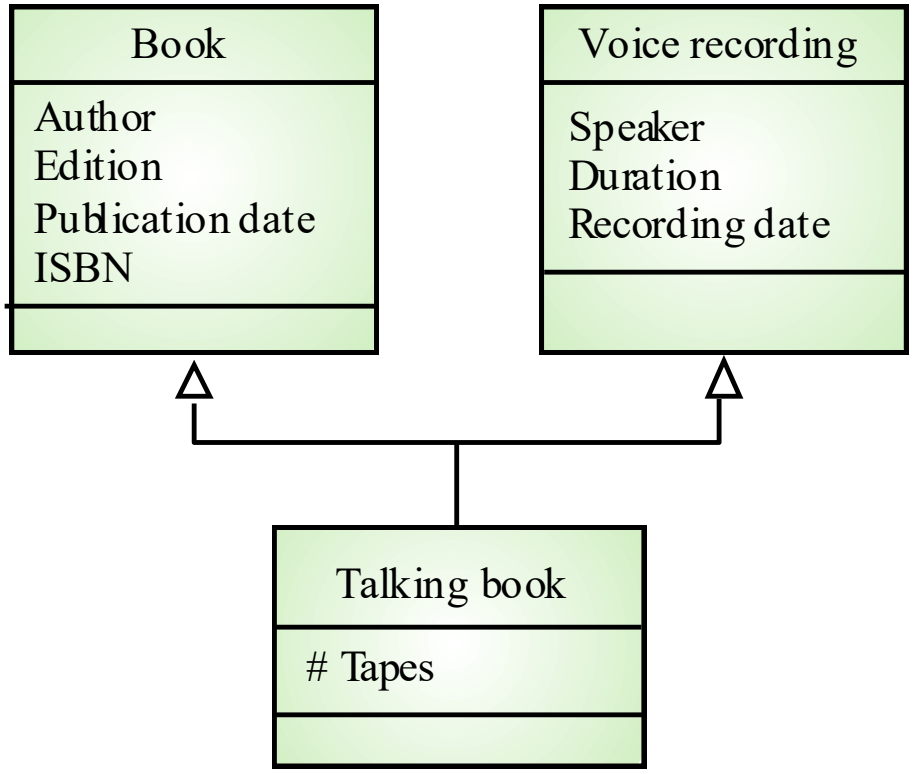
User class hierarchy



Multiple inheritance

- Rather than inheriting the attributes and services from a single parent class, a system which supports multiple inheritance allows object classes to inherit from several super-classes
- Can lead to semantic conflicts where attributes/services with the same name in different super-classes have different semantics
- Makes class hierarchy reorganisation more complex

Multiple inheritance



Advantages of inheritance

- It is an abstraction mechanism which may be used to classify entities
- It is a reuse mechanism at both the design and the programming level
- The inheritance graph is a source of organisational knowledge about domains and systems

Problems with inheritance

- Object classes are not self-contained. They cannot be understood without reference to their super-classes
- Designers have a tendency to reuse the inheritance graph created during analysis. Can lead to significant inefficiency
- The inheritance graphs of analysis, design and implementation have different functions and should be separately maintained

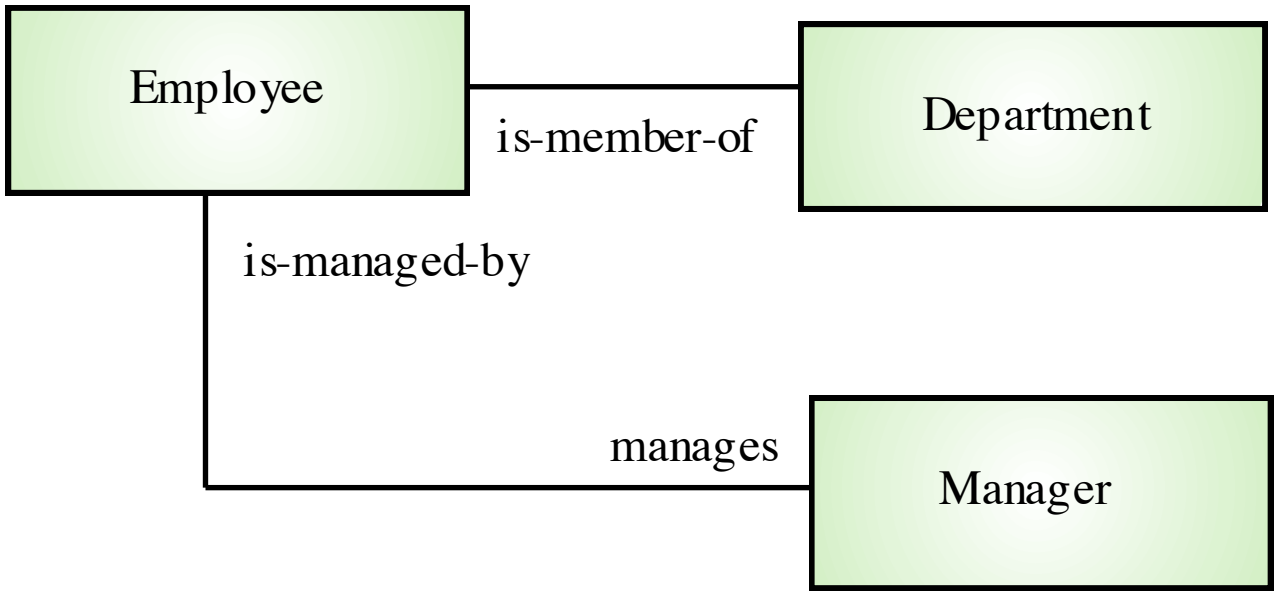
Inheritance and OOD

- There are differing views as to whether inheritance is fundamental to OOD.
  - View 1. Identifying the inheritance hierarchy or network is a fundamental part of object-oriented design. Obviously this can only be implemented using an OOPL.
  - View 2. Inheritance is a useful implementation concept that allows reuse of attribute and operation definitions. Identifying an inheritance hierarchy at the design stage places unnecessary restrictions on the implementation
- Inheritance introduces complexity and this is undesirable, especially in critical systems

Objects Association

- Modeling an association between two classes means that there is some sort of relationship between objects of each class that may be connected.

An association model

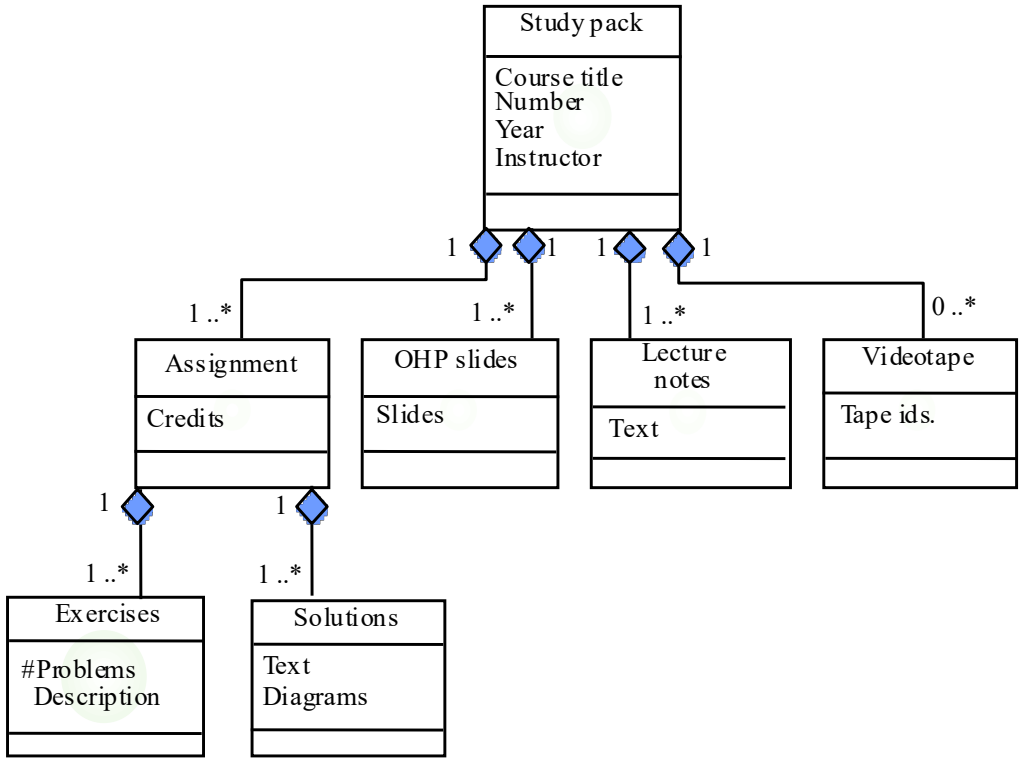


Object aggregation

- Aggregation model shows how classes that are collections are composed of other classes

- Similar to the part-of relationship in semantic data models

Object aggregation

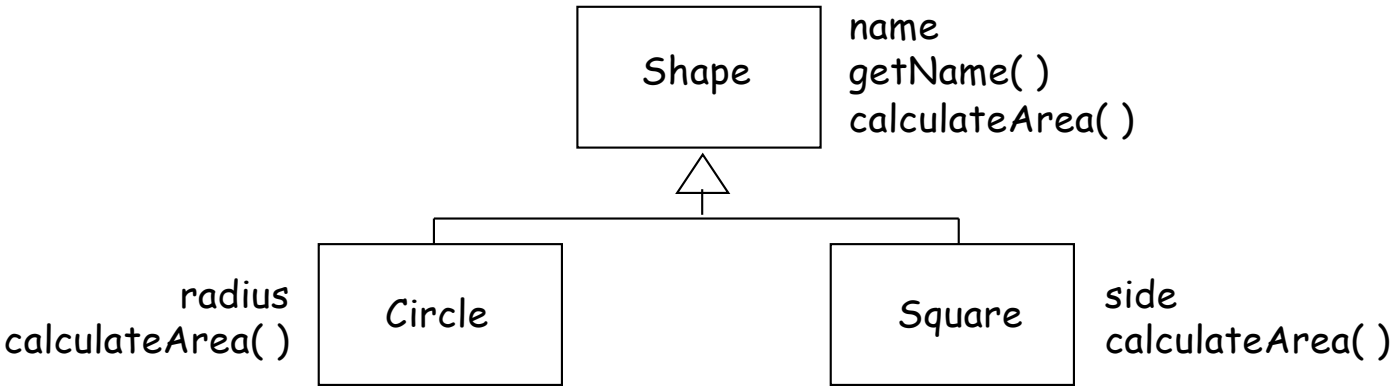


Object Cohesion & Coupling

- Cohesion describes the relationships within modules.
  - Cohesion of a component is a measure of how well it fits together. Each operation provides functionality that allows the attributes of the object to be modified, inspected or used as a basis for service provision.
- Coupling describes the relationships between modules
  - Coupling is an indication of the strength of interconnections between program units. Highly coupled systems have strong interconnections, with program units dependent on each other (shared variables, interchange control function). Loosely coupled system have program units that are independent.

Polymorphism

- the ability of different objects to perform the appropriate method in response to the same message is known as polymorphism.
- the selection of the appropriate method depends on the class used to create the object



Example Polymorphism

```
class Shape {

  private String name;

  public Shape(String aName) { name=aName; }
  public String getName( ) { return name; }
  public float calculateArea( ) { return 0.0f; }

} // End Shape class
```

a generic action

inheritance

```
class Circle extends Shape {
  private float radius;
  public Circle(String aName) { super(aName); radius = 1.0f; }
  public Circle(String aName, float radius) {
    super(aName); this.radius = radius;
  }
  public float calculateArea() { return (float)3.14f*radius*radius; }
} // End Circle class
```

overloading

overriding

```
class Square extends Shape {
    private float side;

    public Square(String aName) {
        super(aName);
        side = 1.0f;
    }

    public Square(String aName, float side) {
        super(aName);
        this.side = side;
    }

    public float calculateArea() {
        return (float) side*side;
    }
} // End Square class
```

Polymorphism Example

```
public class ShapeDemoClient {

  public static void main(String argv[ ]) {

    Shape c1 = new Circle("Circle C1");
    Shape c2 = new Circle("Circle C2", 3.0f);
    Shape s1 = new Square("Square S1");
    Shape s2 = new Square("Square S2", 3.0f);
    Shape shapeArray[ ] = {c1, s1, c2, s2};

    for (int i = 0; i < shapeArray.length; i++) {
      System.out.println("The area of " + shapeArray[i].getName( )
                         + " is " + shapeArray[i].calculateArea( )
                         + " sq. cm.");
    }

  } // End main

} // End ShapeDemoClient1 class
```

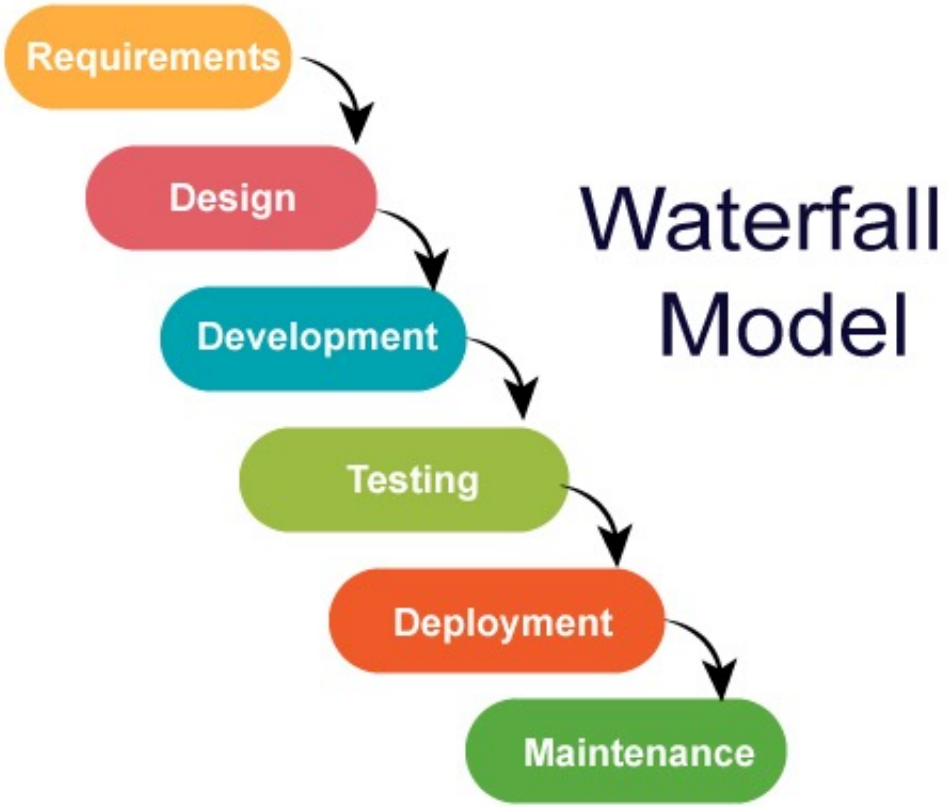
rule of subtype

dynamic binding

OO Analysis and Design

OO Analysis - examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain. In other words, the world (of the system) is modelled in terms of objects and classes.

OO Design - OO decomposition and a notation for depicting models of the system under development. Structures are developed whereby sets of objects collaborate to provide the behaviours that satisfy the requirements of the problem.



Continuous cycles
Small, high-functioning, collaborative teams
Flexible/continuous evolution
Customer involvement

Sequential/linear stages
Upfront planning and in-depth documentation
Works best with well-defined deliverables
Close project manager involvement