CC316: Object Oriented Programming

Lecture 2: Introduction to OOP

Dr. Manal Helal, Fall 2015

http://moodle.manalhelal.com

Previous Lecture

- Introduction to Java Language and the different Editions
- Problem Solving Revision

Lecture Learning Objectives

- In this lesson, you will learn to:
  - State the reasons for the complexity involved in the development of software
  - Define the following terms
    - Objects
    - Classes
    - Messages
    - Methods
  - Explain benefits of the object-oriented approach
  - State the significance of the activities involved in object-oriented analysis and design
  - Create classes in Java

Software Systems Complexity

- Internal Complexity
  - Arises from the composition of a system itself
- External Complexity
  - Arises from the fact that users themselves have only a vague idea of how their system works and have difficulty in expressing their requirements

Complexity Sources

- Difficulty in managing the software development process
- Lack of standards for developing software
- Difficulty in predicting software behaviour.

Simplifying Complexity

- Is done by breaking the system into its component parts and arranging them in a hierarchy

Exercise

- Jane has called a technician to repair her television. How would the technician deal with the complexity of the television?

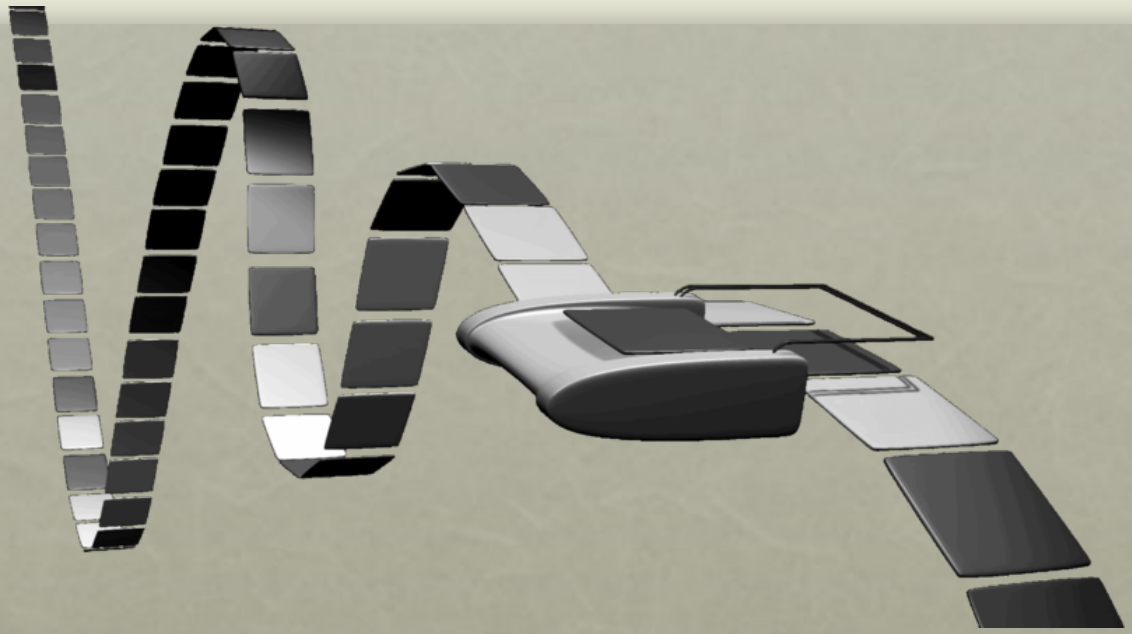
Conflicting Objectives

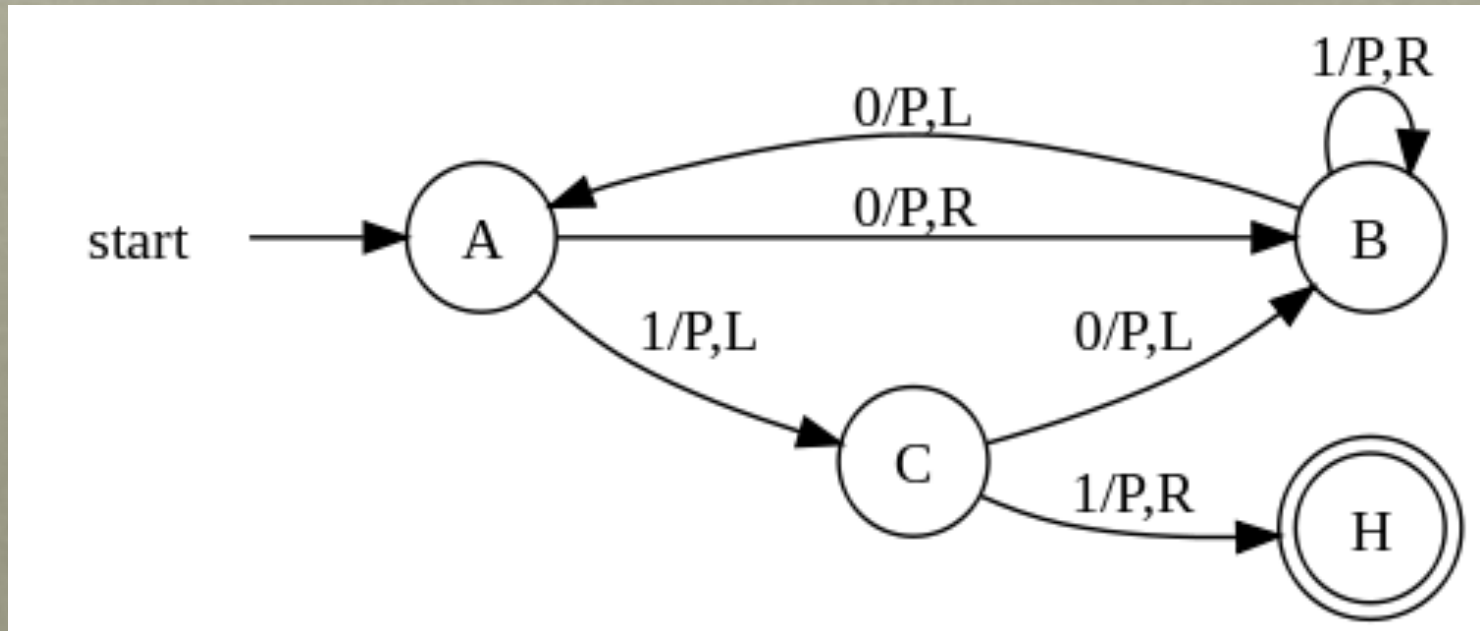
- There are two assertions:
  - OOP is a revolutionary idea, totally unlike anything that has come before in programming languages
  - OOP is an evolutionary step, following naturally on the heels of earlier programming abstractions
- Both are true.

OOP Popularity

- OOP has been the dominant programming paradigm for more than twenty years. Why is it so popular?
  - Proven record of success.
  - Scales well from small problems to large
  - Resonant similarity to techniques for thinking about problems in other domains.
- Nevertheless, programming is still a task that requires skill and learning.

Turing Machine



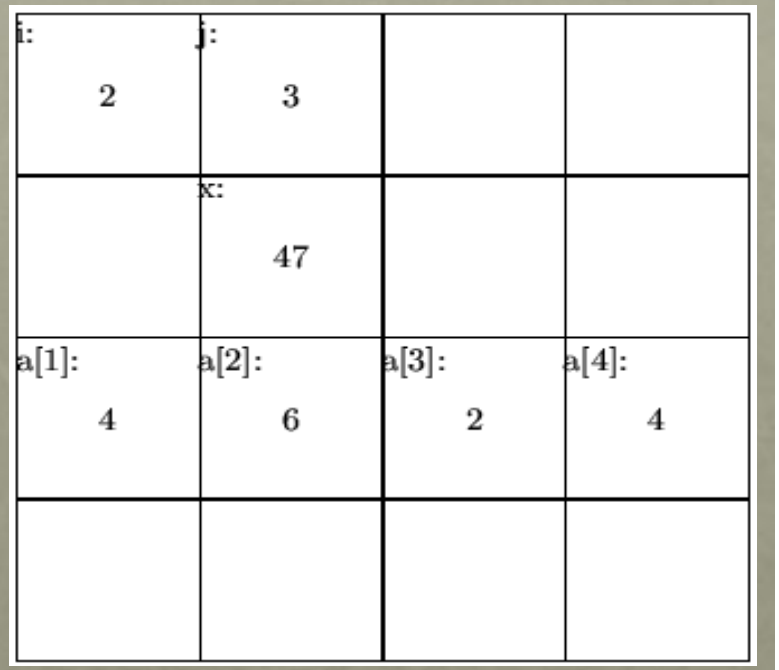


Church's Conjecture

- In computation we have the following assertion:
  - Church's Conjecture: Any computation for which there exists an effective procedure can be realised by a Turing machine language.
- Anything can be done in any language, but it may simply be easier or more efficient to use one language or another.
- Would YOU want to write an event-driven GUI interface in Turing machine? Probably not.
- Bottom line: Languages lead you, but do not prevent you from going anywhere you want

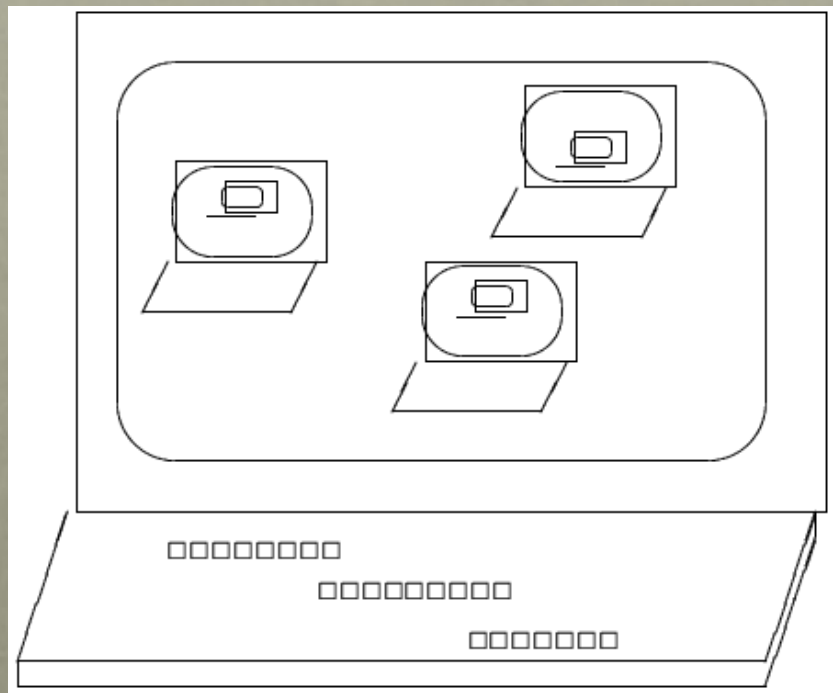
Imperative Programming

- Programming paradigms:
  - Imperative programming is the "traditional" model of computation.
    - State
    - Variables
    - Assignment
    - Loops
  - A processing unit is separate from memory, and "acts" upon memory.
  - Sometimes called the "pigeon-hole" model of computation.



Recursive Design

- Alan Kay thought about this conventional design of the computer, and asked why we constructed the whole out of pieces that were useless by themselves.
- Why not build a whole out of pieces that were similar at all levels of detail? (Think of fractals).
- Idea: A program can be build out of little computing agents.
- The structure of the part mirrors the structure of the larger unit.



Kay's Description of OOP

- Object-oriented programming is based on the principle of recursive design.
  1. Everything is an object
  2. Objects perform computation by making requests of each other through the passing of messages
  3. Every object has it's own memory, which consists of other objects.
  4. Every object is an instance of a class. A class groups similar objects.
  5. The class is the repository for behaviour associated with an object
  6. Classes are organised into singly-rooted tree structure, called an inheritance hierarchy.

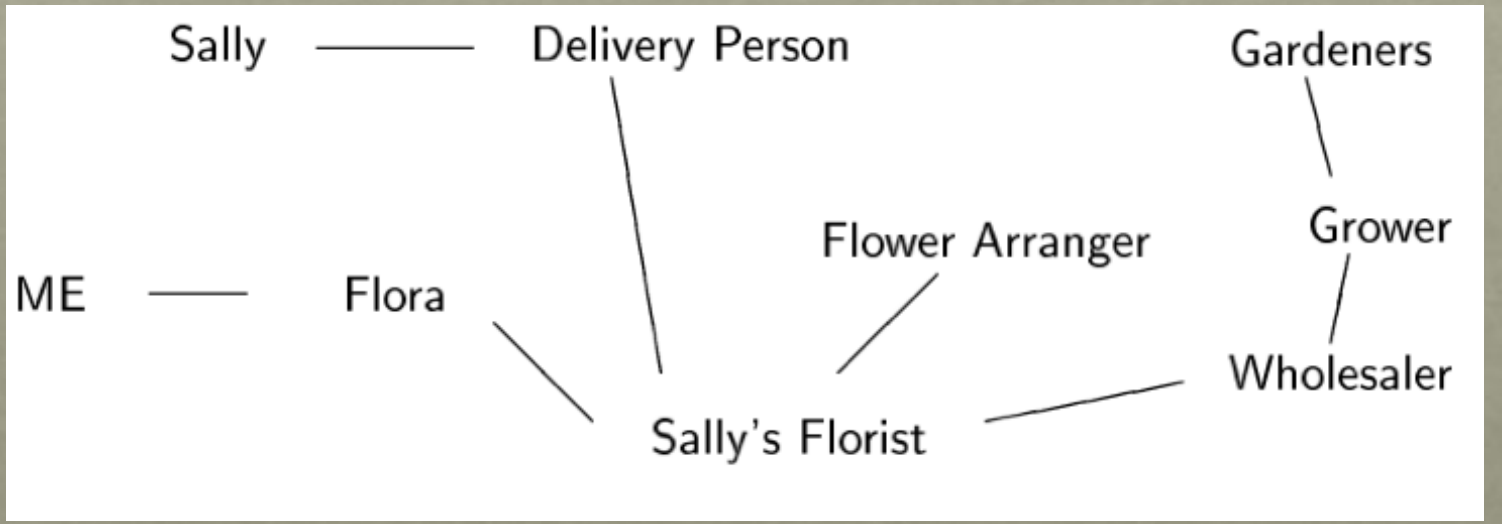
- We can illustrate these principles by considering how I go about solving a problem in real life.

Illustration: Sending Flowers to a Friend

- To illustrate the concepts of OOP in an easily understood framework, consider the problem of sending flowers to a friend who lives in a different city. Chris is sending flowers to Robin.
- Chris can't deliver them directly. So Chris uses the services of the local Florist.
- Chris tells the Florist (named Fred) the address for Robin, how much to spend, and the type of flowers to send.
- Fred contacts a florist in Robins city, who arranges the flowers, then contacts a driver, who delivers the flowers.
- If we start to think about it, there may even be other people involved in this transaction. There is the flower grower, perhaps somebody in charge of arrangements, and so on.

Agents and Communities

- Our first observation is that results are achieved through the interaction of agents, which we will call objects.
- Furthermore, any non-trivial activity requires the interaction of an entire community of objects working together.
- Each object has a part to play, a service they provide to the other members of the community.



Elements of OOP - Object

- **Object:** Is an instance of a class that exhibits some well-defined behaviour
- So we have Kay's first principle.
  1. Everything is an object.
- Actions in OOP are performed by agents, called instances or objects.
- There are many agents working together in my scenario. We have Chris, Robin, the florist, the florist in Robins city, the driver, the flower arranger, and the grower. Each agent has a part to play, and the result is produced when all work together in the solution of a problem.

Characteristics of Objects

- State
  - Is indicated by a set of attributes and the values of these attributes
- Behaviour
  - Is indicated by how an object acts and reacts
- Identity
  - Distinguishes the object from all other objects

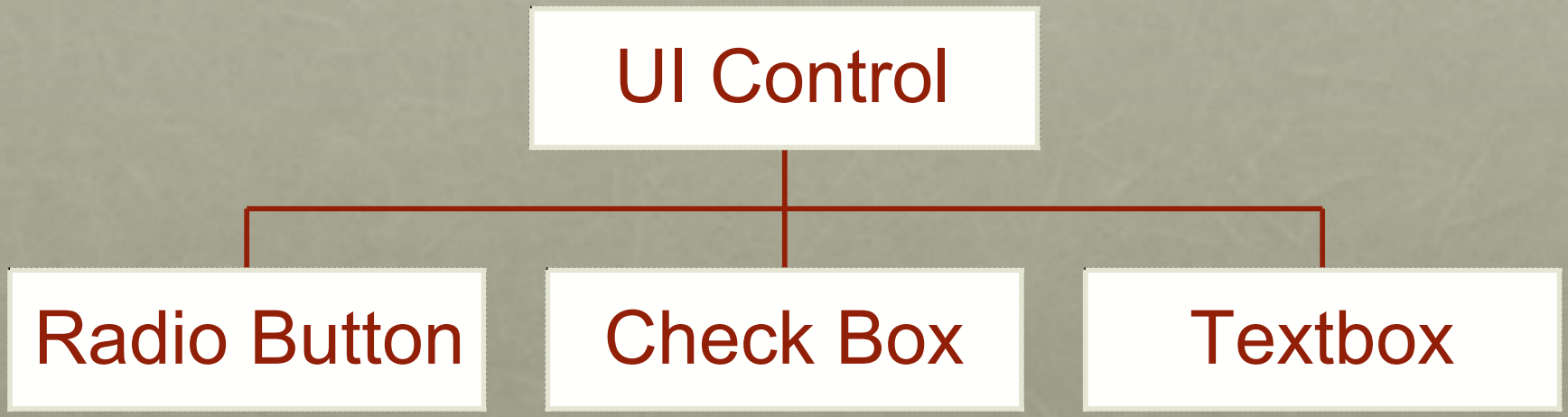
Exercise

- Identify the possible attributes to define the state of the following objects:
  - Tea cup
  - Stereo tape-recorder

Classes

- Define the attributes and behaviours of an object

Example:



Elements of OOP - Messages

- Messages:
  - Are transmitted by one object to another
  - Are transmitted as requests for an action to be taken
  - Are accompanied by additional information needed to carry out the request
- Kay's principle number 2:
  2. Objects perform computation by making requests of each other through the passing of messages
- Actions in OOP are produced in response to requests for actions, called messages. An instance may accept a message, and in return will perform an action and return a value.
- To begin the process of sending the flowers, Chris gives a message to Fred. Fred in turn gives a message to the florist in Robins city, who gives another message to the driver, and so on.

Information Hiding

- Notice that I, as a user of a service being provided by an object, need only know the name of the messages that the object will accept.
- I need not have any idea how the actions performed in response to my request will be carried out.
- Having accepted a message, an object is responsible for carrying it out.

Elements of OOP - Receivers

- Messages differ from traditional function calls in two very important respects:
  - In a message there is a designated receiver that accepts the message
  - The interpretation of the message may be different, depending upon the receiver

Different Receivers, Same Message, Different Actions

```
var
        Fred : Florist;
        Elizabeth : Friend;
        Ken : Dentist;

begin
        Fred.sendFlowersTo(myFriend);  { will work }
        Elizabeth.sendFlowersTo(myFriend); { will also
work }
        Ken.sendFlowersTo(myFriend); { will probably not
work }
end;
```

- The same message will result in different actions, depending upon who it is given to.

Behaviour and Interpretation

- Although different objects may accept the same message, the actions (behaviour) the object will perform will likely be different.
- The determination of what behaviour to perform may be made at run-time, a form of late binding.
- The fact that the same name can mean two entirely different operations is one form of polymorphism, a topic we will discuss at length in subsequent chapters.

Elements of OOP - Recursive Design

- Kay’s Third Principal:
  3. Every object has it's own memory, which consists of other objects.
- Each object is like a miniature computer itself - a specialised processor performing a specific task.

Non-interference

- It is important that objects be allowed to perform their task however they see fit, without unnecessary interactions or interference with other objects.
  - “Instead of a bit-grinding processor raping and plundering data structures, we have a universe of well-behaved objects that courteously ask each other to carry out their various desires” -- Dan Ingalls.
  - “Ask not what you can do to your data structures, but ask what your data structures can do for you”

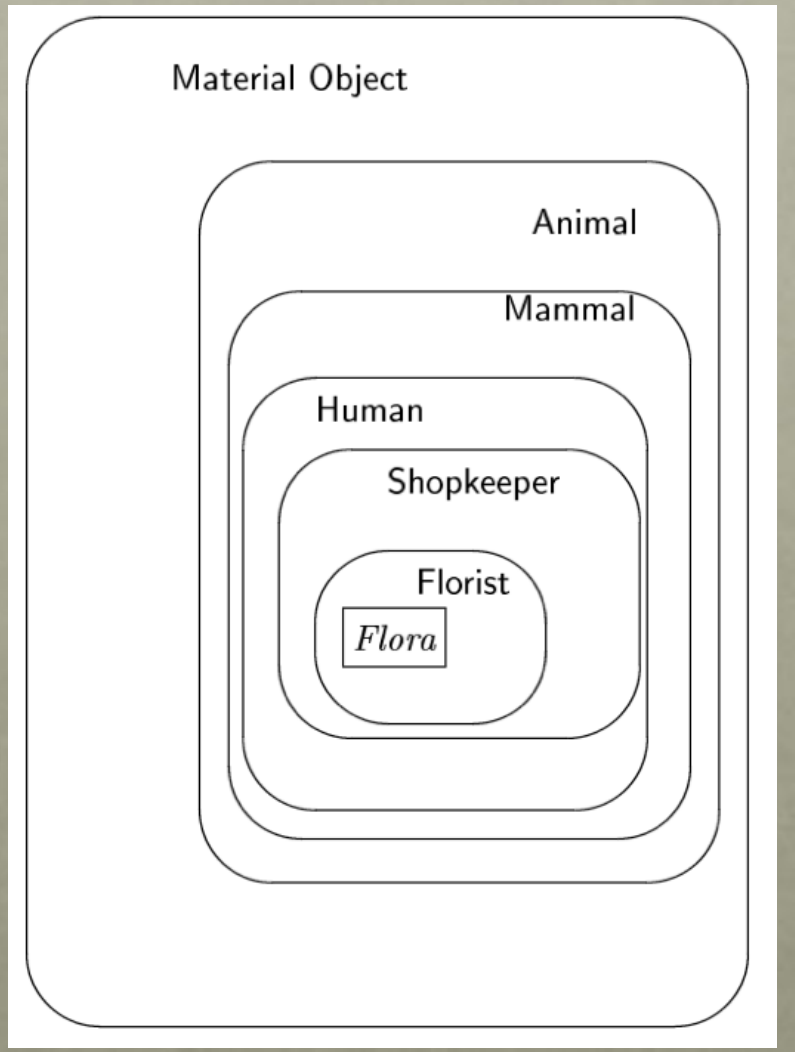
Elements of OOP - Classes

- Kay's Fourth and Fifth Principals:
  4. Every object is an instance of a class. A class groups similar objects.
  5. The class is the repository for behaviour associated with an object.
- The behaviour I expect from Fred is determined from a general idea I have of the behaviour of Florists.
- We say Fred is an instance of the class Florist.
- Behaviour is associated with classes, not with individual instances. All objects that are instances of a class use the same method in response to similar messages.

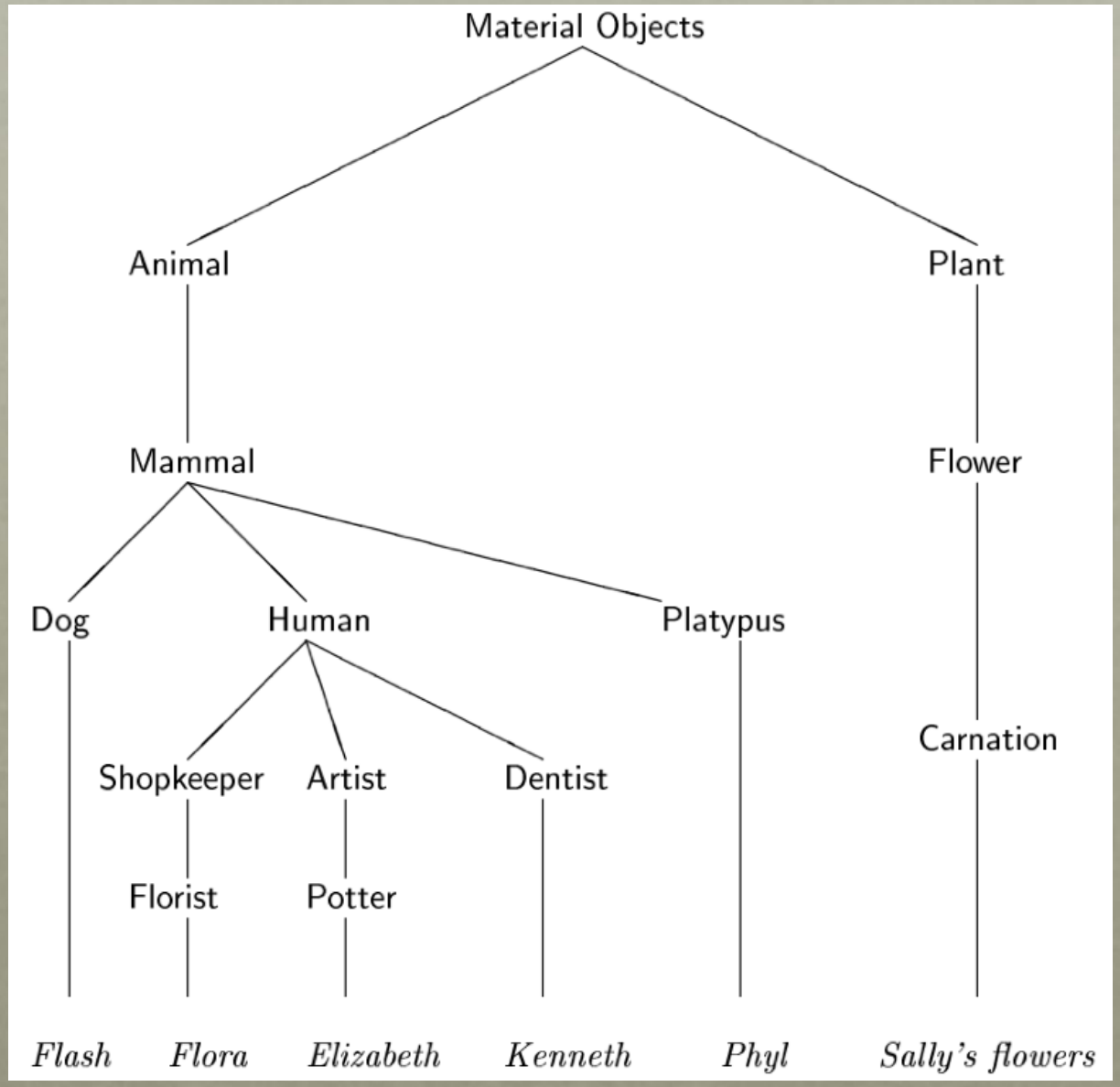
Hierarchies of Categories

- But there is more that I know about Fred then just that he is a Florist. I know he is a Shop-Keeper, and a Human, and a Mammal, and an Animal, and a Material Objects, and so on.

- At each level of abstraction I have certain information recorded. That information is applicable to all lower (more specialised) levels.



Class Hierarchies



Elements of OOP - Inheritance

- Kay's Sixth Principal:
  6. Classes are organised into a singly-rooted tree structure, called an inheritance hierarchy
- Information (data and/or behaviour) I associate with one level of abstraction in a class hierarchy is automatically applicable to lower levels of the hierarchy.

Elements of OOP - Methods

- Are a set of actions taken by the receiver object in response to the request

Elements of OOP - Overriding

- Subclasses can alter or override information inherited from parent classes:
  - All mammals give birth to live young
  - A platypus is an egg-laying mammal
- Inheritance combined with overriding are where most of the power of OO originates.

A platypus is a curious animal - it is considered to be a Mammal but it nevertheless reproduces by laying eggs.

Computing as Simulation

- The OOP view of computation is similar to creating a universe of interacting computing objects
- Similar to the way in which a committee or club might be organised
- Also very similar to a style of simulation called discrete event-driven simulation
- Easily under estimated advantage of this view -- power of metaphor.

Metaphor and Problem Solving

- Because the OOP view is similar to the way in which people go about solving problems in real life (finding another agent to do the real work!), intuition, ideas, and understanding from everyday experience can be brought to bear on computing.

- On the other hand, common sense was seldom useful when computers were viewed in the process-state model, since few people solve their everyday problems using pigeon-holes.

Exercise

- Dr. James and Mr. Hyde went to the railway station to book two tickets in the Flying express for 3rd December by AC 1st class. Identify the following:

a. The possible receiver of the message in this situation

b. The possible method that the receiver can use

Benefits of the Object-Oriented Approach

- Realistic modelling
  - Easy to use
- Reusability
  - Saves time and cost

Exercise

- State whether the following situations demonstrate reusability:

a. Recycling paper

b. Pump reusability (same pump is used in a well and in a fuel station)

Benefits of Object-Oriented Approach (Cont'd).

- Resilience to change
- Easy to maintain
- Parts of the system can be refined without any major change in other parts

Object-Oriented Analysis (OOA)

- Analysis:
  - Is a phase where users and developers get together and arrive at a common understanding of the system
  - Requires the developer to concentrate on obtaining maximum possible information about the problem domain
  - Results in one of the end products as specification of the function of the system

Object-Oriented Design (OOD)

- Design:
  - Generates the blueprint of the system that has to be implemented
  - Involves identifying classes using:
    - Abbott's technique

Object-Oriented Design (OOD) (Contd.)

- Abbott's technique follows the listed steps:
  - Write English description of the problem
  - Underline nouns (nouns represent candidate classes)

Object-Oriented Programming (OOP)

- Is a way of writing programs
- Some applications built using OOP techniques are:
  - Computer-Aided Design (CAD)
  - Computer-Aided Manufacturing (CAM)
  - Artificial Intelligence (AI) and Expert Systems
  - Object-Oriented Databases

Generations of Computer Languages

- First generation:
  - Is a machine-level programming language, no compiler or assembler; entered using the front panel switches of the computer.
- Second generation:
  - Can be read and written by a programmer; and must be converted to a machine readable form to run on a computer (assembler).
  - C kernel function and human fine tuning to the machine.
- Third generation (High Level Languages) – 1950s:
  - Brought logical structure to software, programmer friendly, human readable natural language and block structure, improved support for aggregate data types, and expressing concepts in a way that favours the programmer, not the computer.
  - Late 1950s, Fortran, ALGOL, Ada, and COBOL, and most Popular: C, C++, C#, Java, BASIC and Pascal.
- Fourth Generation - 1970s-1990:
  - Packages of systems development software including very high level programming languages and development environments providing higher abstraction and statement power.
  - Components: 'Analyst Workbench' designed with a central data dictionary system, a library of loosely coupled design patterns, a CRUD generator, report generator, end-user query language, DBMS, visual design tool and integration API
  - Rapid Application Development (RAD) tools are more oriented toward problem solving and systems engineering.
  - Examples: Power Builder, Mathematica, Matlab, R, SAS, SPSS,
- Fifth Generation – 1980s:
  - Based on solving problems using constraints given to the program, rather than using an algorithm written by a programmer, mainly in AI research
  - Examples: Prolog, OPS5, and Mercury.

Evolution of C++ as an OOP Language

- In the early 1980s, Bjarne Stroustrup developed the C++ language
- C++ was originally known as 'C with classes’

Exercise: Problem Statement

- As a member of a team that is developing the billing system software for Diaz Telecommunications Inc., you have been assigned the task of creating a software module that accepts and displays customer details. Declare the Customer class and the member functions. The member function to accept customer details should display the message “Accepting Customer Details”. Similarly, the member function to display customer details on the screen should display the message “Displaying Customer Details.”

Solution

```
import java.io.*;
class customer {
  static String name;  static String address;
  public static void readValues() {
    try {
      BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Please Enter Customer Name:");
      name = in.readLine();
      System.out.println("Please Enter Customer Address:");
      address = in.readLine();
      }
      catch (java.io.IOException ioe) {
        System.out.println("IO Error: " + ioe.getMessage());
        System.exit(1);
      }
    }
  public static void displayValues() {
    System.out.println("Customer Name: "+ name);
    System.out.println("Customer Name:" + address);
  }
  public static void main (String[] args) {
    readValues ();
    displayValues ();
  }
}
```

Exercise

- As a member of a team that is developing an automated booking system for the Railways, you have been assigned the task of creating a module that accepts the details of a passenger and checks whether the ticket has been confirmed or is in the waiting list. The module then prints the list of confirmed passengers. Declare a class Ticket, which consists of three member functions, booking(), status(), and print().

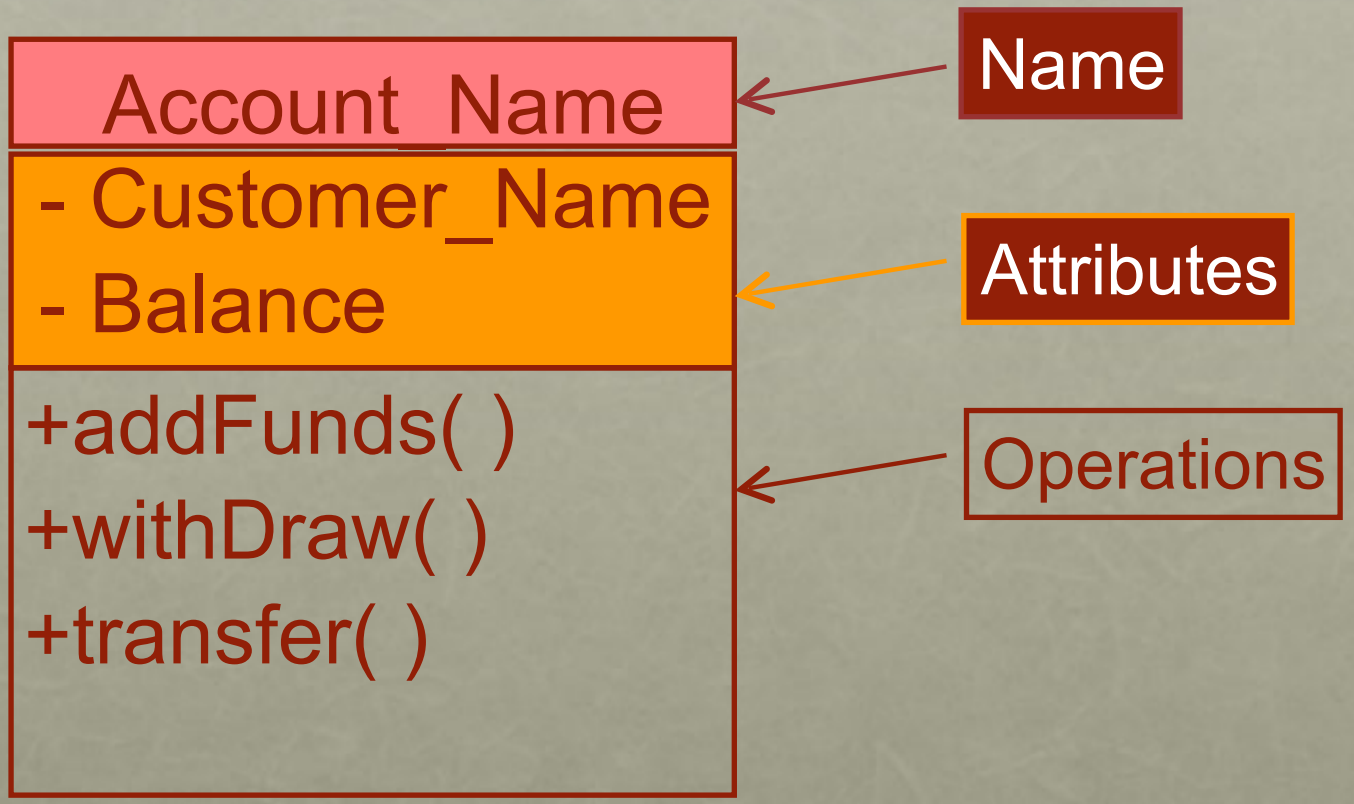
UML - Class diagram

- Used for describing structure and behaviour in the use cases
- Provide a conceptual model of the system in terms of entities and their relationships
- Used for requirement capture, end-user interaction
- Detailed class diagrams are used for developers

UML - Class Representation

- Each class is represented by a rectangle subdivided into three compartments
  - Name
  - Attributes
  - Operations
- Modifiers are used to indicate visibility of attributes and operations.
  - '+' is used to denote *Public* visibility (everyone)
  - '#' is used to denote *Protected* visibility (friends and derived)
  - '-' is used to denote *Private* visibility (no one)
- By default, attributes are hidden and operations are visible.

UML An example of Class Diagram



Assignment – ass2

- Design a UML Class diagram to model inheritance relationships between a vehicle class, a bicycle class, a motor vehicle class, a car class, and a motor bike class with the following specifications:
  - All Vehicles have some common attributes of speed, colour, and common behaviour to turnLeft and turnRight.
  - Bicycle and MotorVehicle are both kinds of Vehicle.
  - Bicycle has an additional behavious to ring bell.
  - MotorVehicles have size of engines and license plates attributes.
  - MotorBike and Car are considered types of MotorVehicles.
  - MotorBike has an additional behaviour to generate rev Engine.
  - Car has an additional attribute of number of doors and behaviour to switch on air condition.
  - Include the behaviour that allows us to examine all attributes in all classes.
- Due next week.