BBM 102 – Introduction to Programming II

Introduction to
Object Oriented Programming

Today

- **Software as a Complex Thing**
  - Dealing with Complexity
  - Functional Decomposition
  - Structured Programming
- **Object Oriented Paradigm**
  - Principles of Object Orientation
  - Classes and Objects
  - Sample Object Designs

Today

- **Software as a Complex Thing**
  - Dealing with Complexity
  - Functional Decomposition
  - Structured Programming
- **Object Oriented Paradigm**
  - Principles of Object Orientation
  - Classes and Objects
  - Sample Object Designs

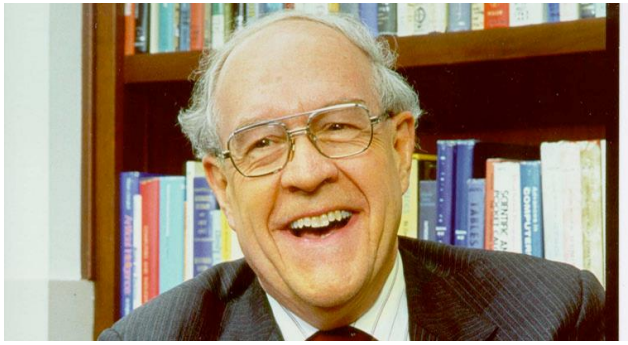
Software in Modern World

- We can't run the modern world without software.
  - National infrastructures and utilities are controlled by computer-based systems
  - Most electrical products include a computer and controlling software
  - Industrial manufacturing and distribution is completely computerized, as is the financial system.
  - Entertainment, including the music industry, computer games, and film and television, is software intensive

Software in Modern World: Examples

Software is a Complex Thing

- Software development is a serious business.
- We are not talking about a program that finds *N Fibonacci* numbers anymore.
- We are talking about *industrial-strength (professional)* software.
  - It is intensely difficult, if not impossible, for the individual developer to comprehend all the subtleties of its design.
- "The complexity of software is an essential property, not an accidental one." -- F.P. Brooks

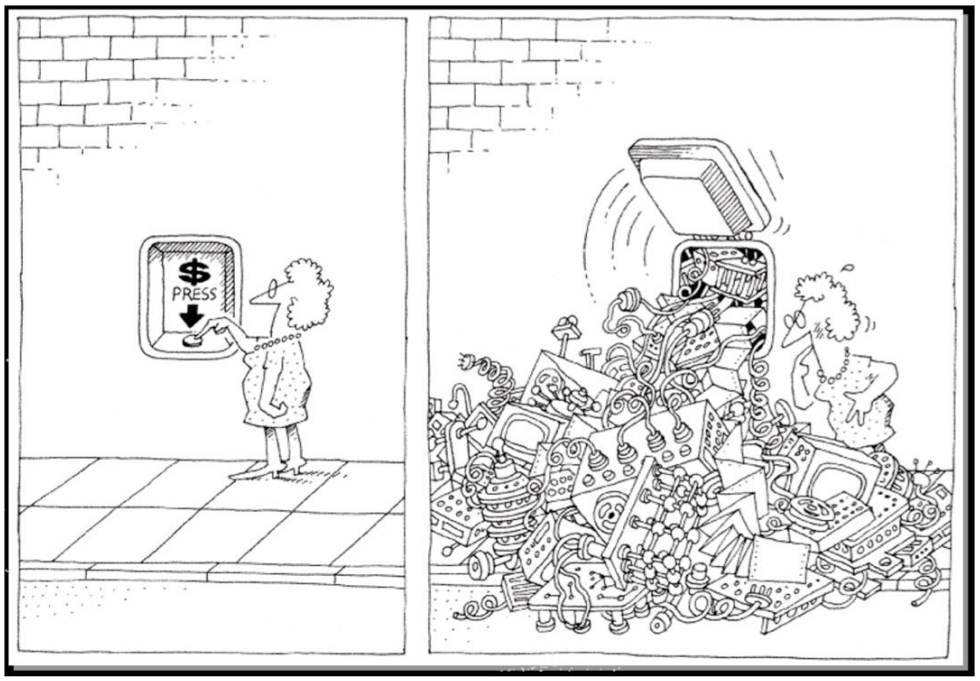


Professional Software Development

- Professional software, intended for use by someone apart from its developer, is usually *developed by teams* rather than individuals.
  - It is maintained and changed throughout its life.

- Software development is referred to mean *professional software development*, rather than individual programming.
  - It includes techniques that support program specification, design, and evolution, none of which are normally relevant for personal software development.

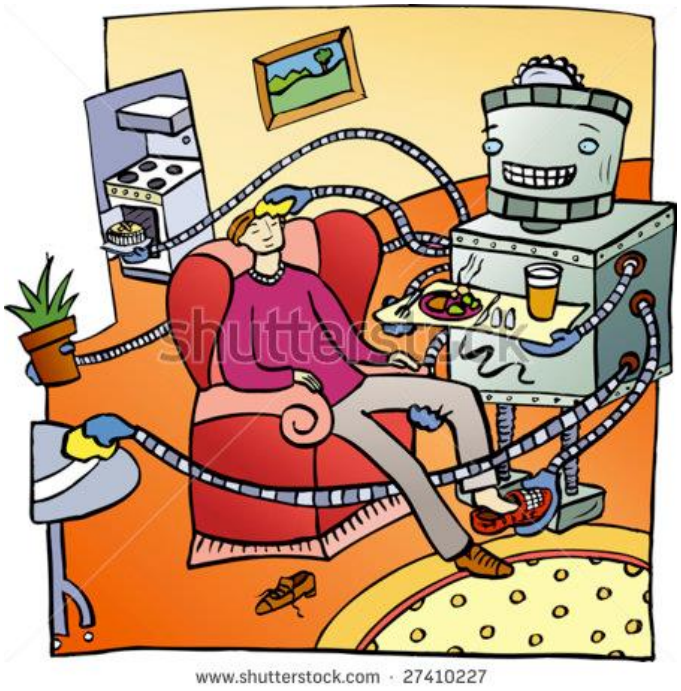
Inherent Complexity of Software



- The complexity of the problem domain
- The difficulty of managing the development process
- The flexibility possible through software
- The problem of characterizing the behavior of discrete systems

The Complexity of the Problem Domain

- The problem often involves elements of *inescapable complexity*, in which we find a myriad of competing, perhaps even contradictory**, requirements**.
  - Consider the requirements for an autonomous robot. Its raw functionality is difficult enough to comprehend.
  - Also consider **nonfunctional requirements** such as usability, performance, survivability, and reliability.

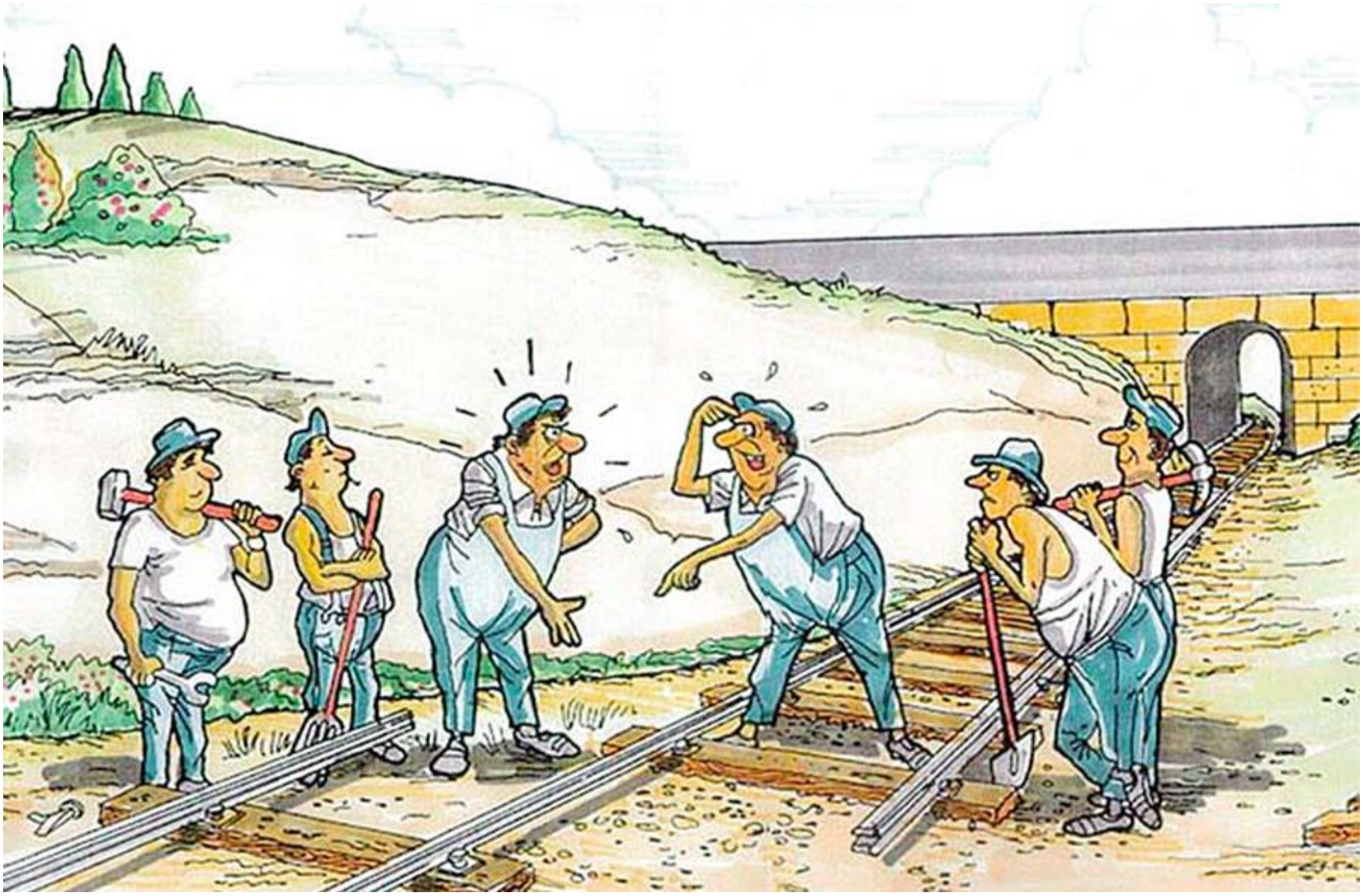


- The requirements of a software system often change during its development.

The Difficulty of Managing the Development Process

- Industrial-strength software typically consists of hundreds and sometimes thousands of **separate modules**.
- No one person can ever understand such a system completely.
- This amount of work demands that we use a **team of developers**, and ideally we use as small a team as possible.
- There are challenges associated with team development.
  - Having more developers means **more complex communication** and more **difficult coordination**, particularly if the team is geographically dispersed.
  - With a team of developers, the key management challenge is always to maintain a unity and integrity of design.

The Difficulty of Managing the Development Process



Software (Development) Process

The systematic approach to software development is called a software process which is defind as a sequence of activities that leads to the production of a software product:

- Software specification, where customers and engineers define the software that is to be produced and the constraints on its operation.
- Software development, where the software is designed and programmed.
- Software validation, where the software is checked to ensure that it is what the customer requires.
- Software evolution, where the software is modified to reflect changing customer and market requirements.

Software (Development) Processes

- Different types of systems need different development processes.
  - For example, *real-time software* in an aircraft has to be completely specified before development begins.
  - In *e-commerce systems*, the specification and the program are usually developed together.

- Consequently, these generic activities may be organized in different ways and described at different levels of detail depending on the type of software being developed.
  - There are many different types of software. There is no universal software engineering method or technique that is applicable for all of these.

The Flexibility Possible through Software

- Software is *abstract* and *changeable*.

- A home-building company generally does not operate its own tree farm from which to harvest trees for construction. In the software industry such practice is common.
  - Software offers the ultimate flexibility, so it is possible for a developer to express almost any kind of **abstraction**.
  - This flexibility forces the developer to craft virtually all the primitive building blocks on which these higher-level abstractions stand.
  - While the construction industry has uniform building codes and standards for the quality of raw materials, few such standards exist in the software industry due to *abstactness of the software*.

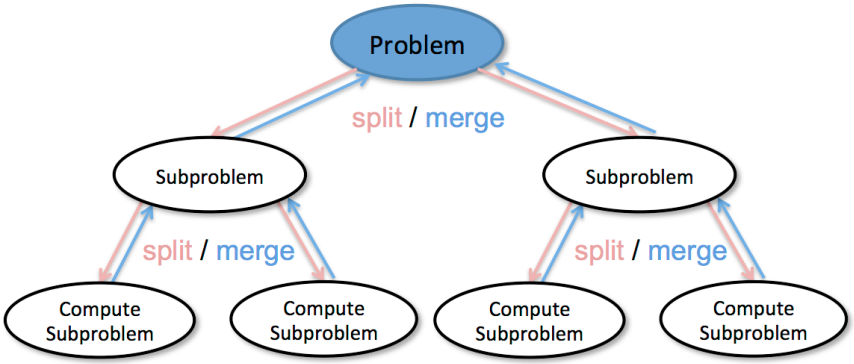
Characterizing the Behavior of Discrete Systems - Difficulties

- There may be hundreds or thousands of variables as well as more than one thread of control.
  - The entire collection of these variables, their current values, and the current address and calling stack of each process within the system constitute the **present state** of the application.

- Each event external to a software system has the potential of placing that system in a **new state**
  - The mapping from state to state is not always deterministic.
  - The event may **corrupt the state of a system** because its designers failed to take into account certain interactions among events.

- **Vigorous testing** is essential but not possible to model the complete behavior of large discrete systems.
  - We must survive with acceptable levels of confidence.

How to Deal with Complexity?



The technique of mastering complexity has been known since ancient times: *divide et impera* (divide and rule)

- When designing a complex software system, it is essential to decompose it into smaller and smaller parts, each of which we may then refine independently.

- In this manner, we satisfy the very real constraint that exists on the channel capacity of human cognition: ***To understand any given level of a system, we need only comprehend a few parts (rather than all parts) at once.***

How Did You Decompose So Far?

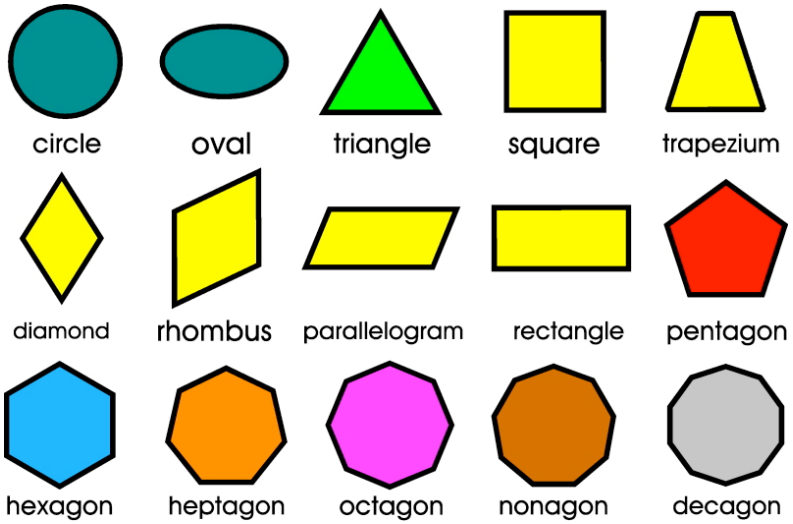
- Functional Decomposition
  - A natural way to deal with complexity
  - Break down (decompose) the problem into the **functional steps** that compose it.

- Example: Write a code to
  - Access a description of shapes that were stored in a database
  - Display those shapes

  - *How do you decompose these tasks?*

Functional Decomposition - Example

- It would be natural to think in terms of the steps required:
  1. Locate the list of shapes in the database.
  2. Open up the list of shapes.
  3. Sort the list according to some rules.
  4. Display the individual shapes on the monitor.



Functional Decomposition - Example (cont'd)

- Further breaking down of Step 4 is possible:
  1. Locate the list of shapes in the database.
  2. Open up the list of shapes.
  3. Sort the list according to some rules.
  4. Display the individual shapes on the monitor.
     a) Identify the type of shape.
     b) Get the location of shape.
     c) Call the appropriate function that will display the shape by giving the shape's location.

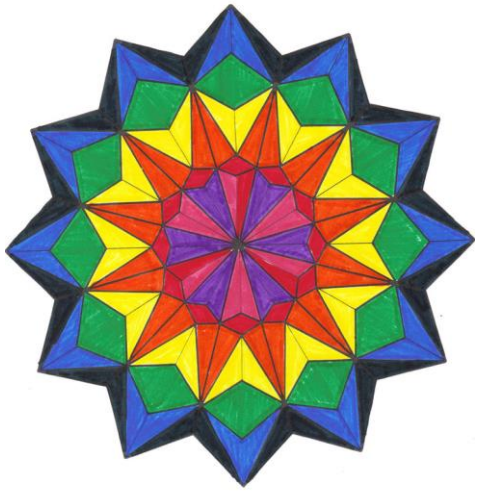
We will revisit this algorithm (Step 4.c more precisely) few slides later.

Is Functional Decomposition Perfect?

- It does not help us much prepare the code for possible changes in the future.

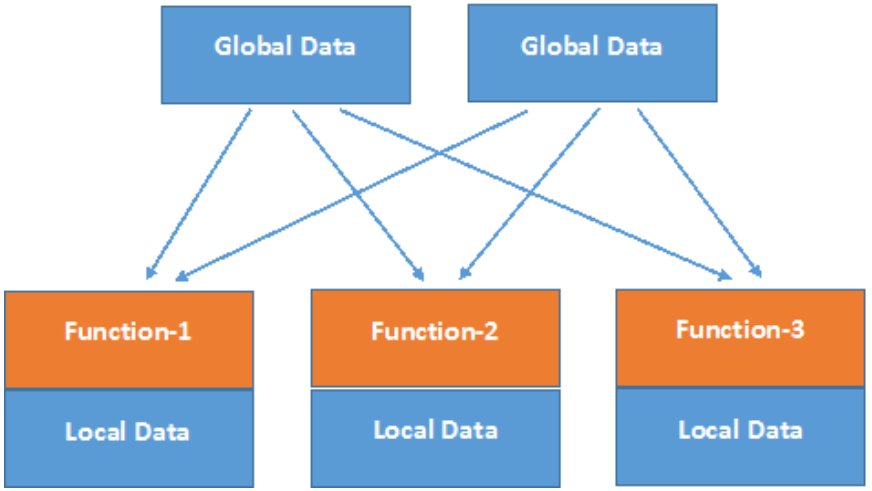
- Many bugs originate with changes to the code.
  - Change creates opportunities for mistakes and unintended consequences.

- Nothing you can do will stop change.
  - You can never get all of requirements from the user.
  - When the business domain changes, so the software.
  - Future is unknown – things will change.

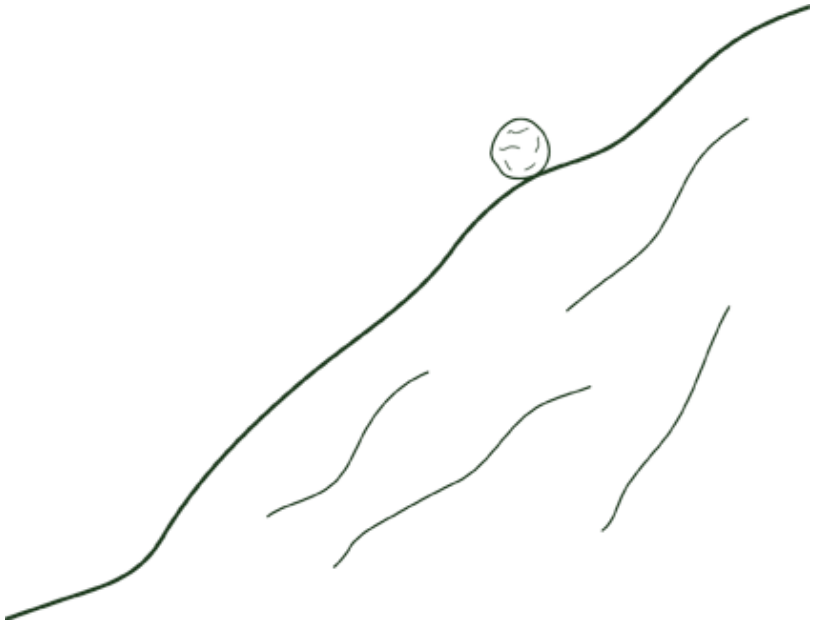


Unwanted Side Effect

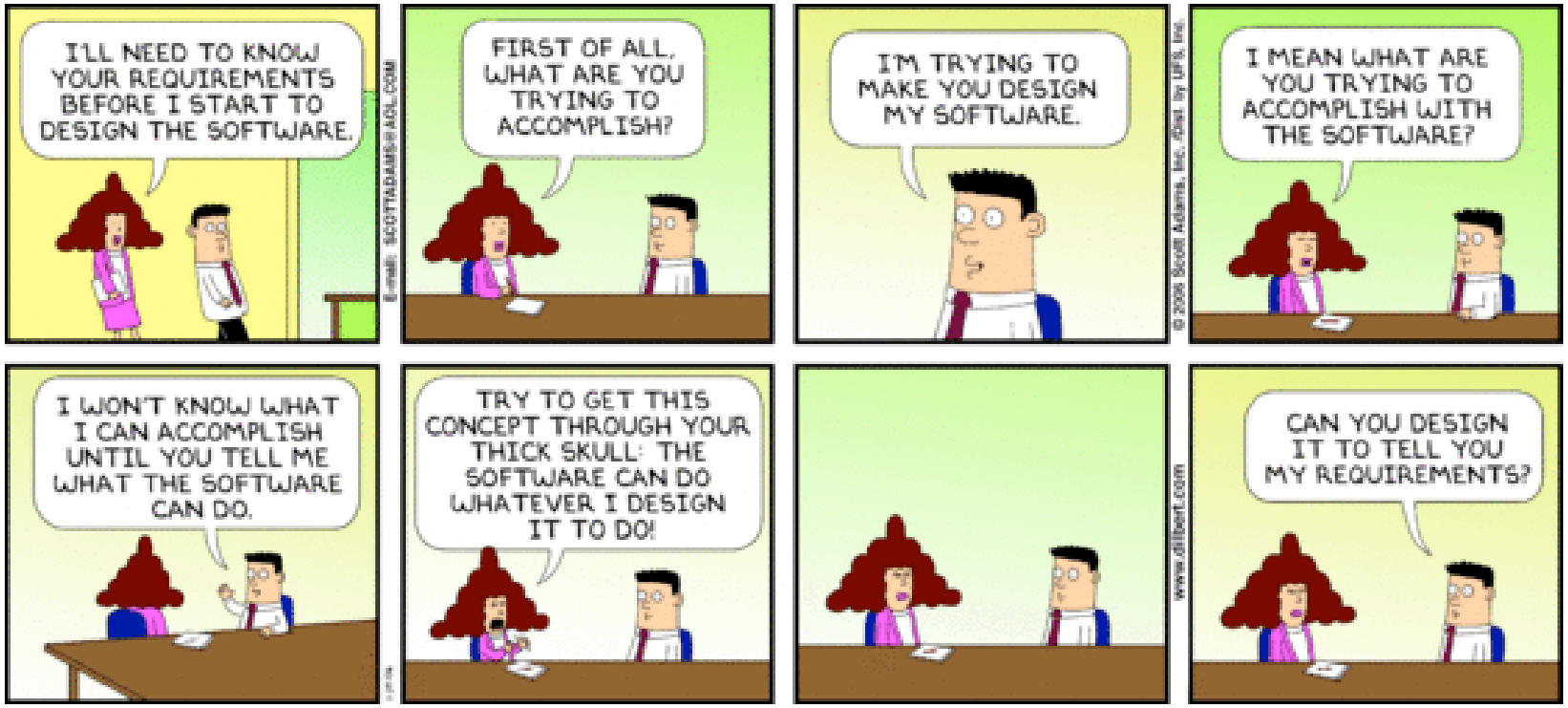
- Make a change to a function or a piece of data in one area of the code.
  - Then, have an unexpected impact on other pieces of code.



- Wrong focus: Changes to one set of functions or data impact other sets of functions and other sets of data.
  - Like a snowball that picks up snow as it rolls downhill!



Problem with Requirements



Problem with Requirements

- **Requirements** from users are
  - incomplete, usually wrong, misleading, not telling the whole story.

- **Requirements change** for a very simple set of reasons:
  - The users see new possibilities for the software after discussions with developers.
  - Developers become more familiar with users' problem domain.
  - The environment in which the software is being developed changes.

- You must write your code to accommodate change
  - Not give up on **gathering good requirements**

Let's Take a Close Look on Step 4.c

- Step 4.c of displaying shapes
  - Call appropriate function that will display shape

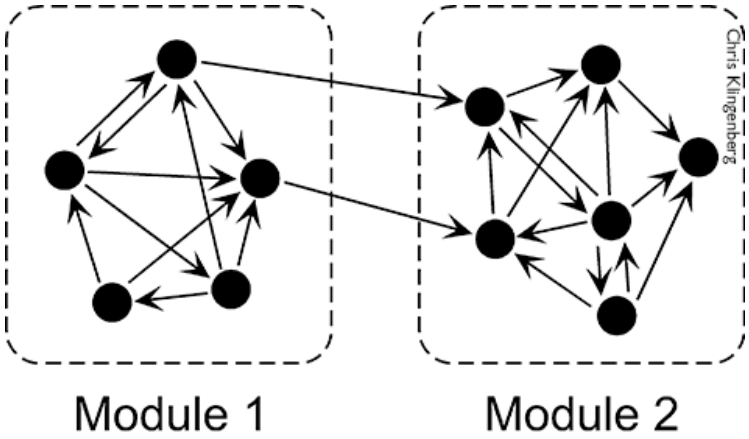
```
function: display shape
input: type of shape, description of shape
action: switch (type of shape)
                case square:
                        put display function for square here
                case circle:
                        put display function for circle here
```

- Use **modularity** to contain variation

Modularity

- May or may not be possible to have a consistent description of shapes that will work for all shapes

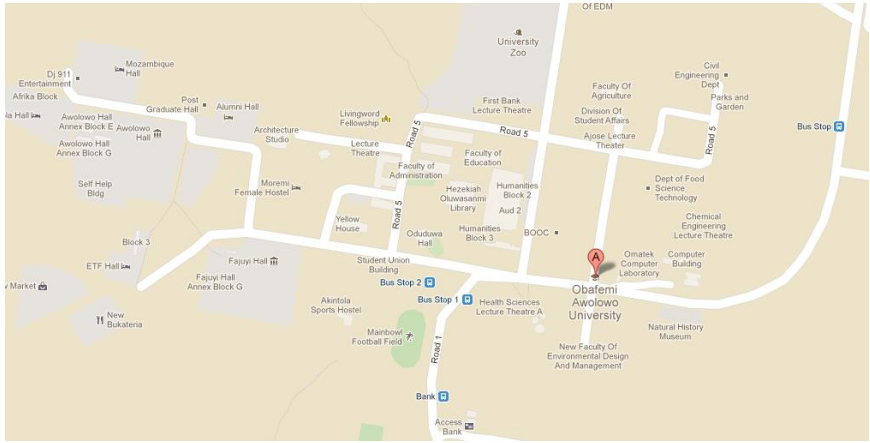
- Modularity
  - Makes the code more understandable
  - Understandability makes the code easier to maintain
  - Does not deal with all of the variation it might encounter



- The goal is to create routines with
  - internal integrity (*strong cohesion* - how closely the operations in a routine are related) and
  - small, direct, visible, and flexible relations to other routines (*loose coupling* -connection between routines)

Structured Programming Approach Example

- Students in your class are having another class. They are going to attend to another class, but they do not know where their next class is located.



- Algorithm:
  1. Get a list of students in the class.
  2. For each student on this list:
     a) Find the next class she/he is taking
     b) Find the location of that class
     c) Find the way to get from your classroom to the student's next class
     d) Tell the student how to get to their next class

- Would you actually follow this approach?
  - Post directions in the back of the room
  - Expect everyone would know what their next class was

What is the Difference?

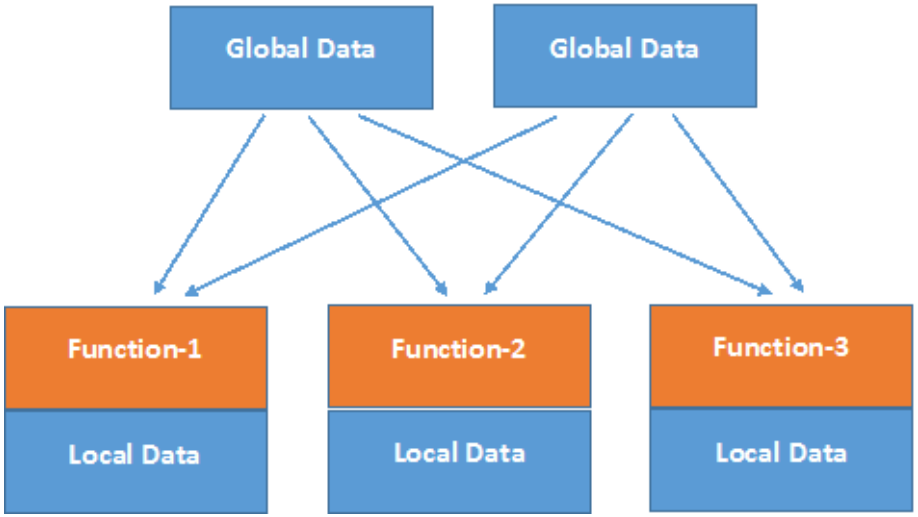
- First case
  - Give explicit directions to everyone.
  - No one other than you is responsible for anything.

- Second case
  - Give general instructions
  - Each person figure out how to do the task himself or herself

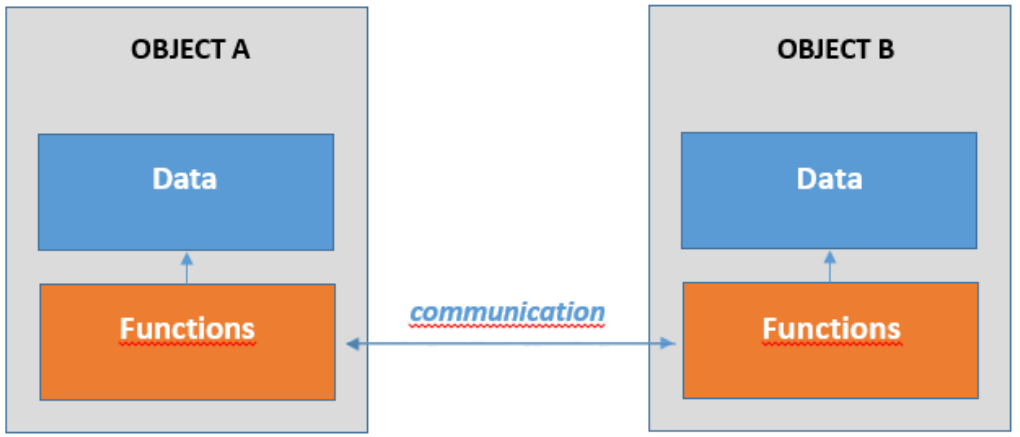
- Shift of responsibility may not be a bad thing

Srtuctured versus Object-Oriented Programming Constructs

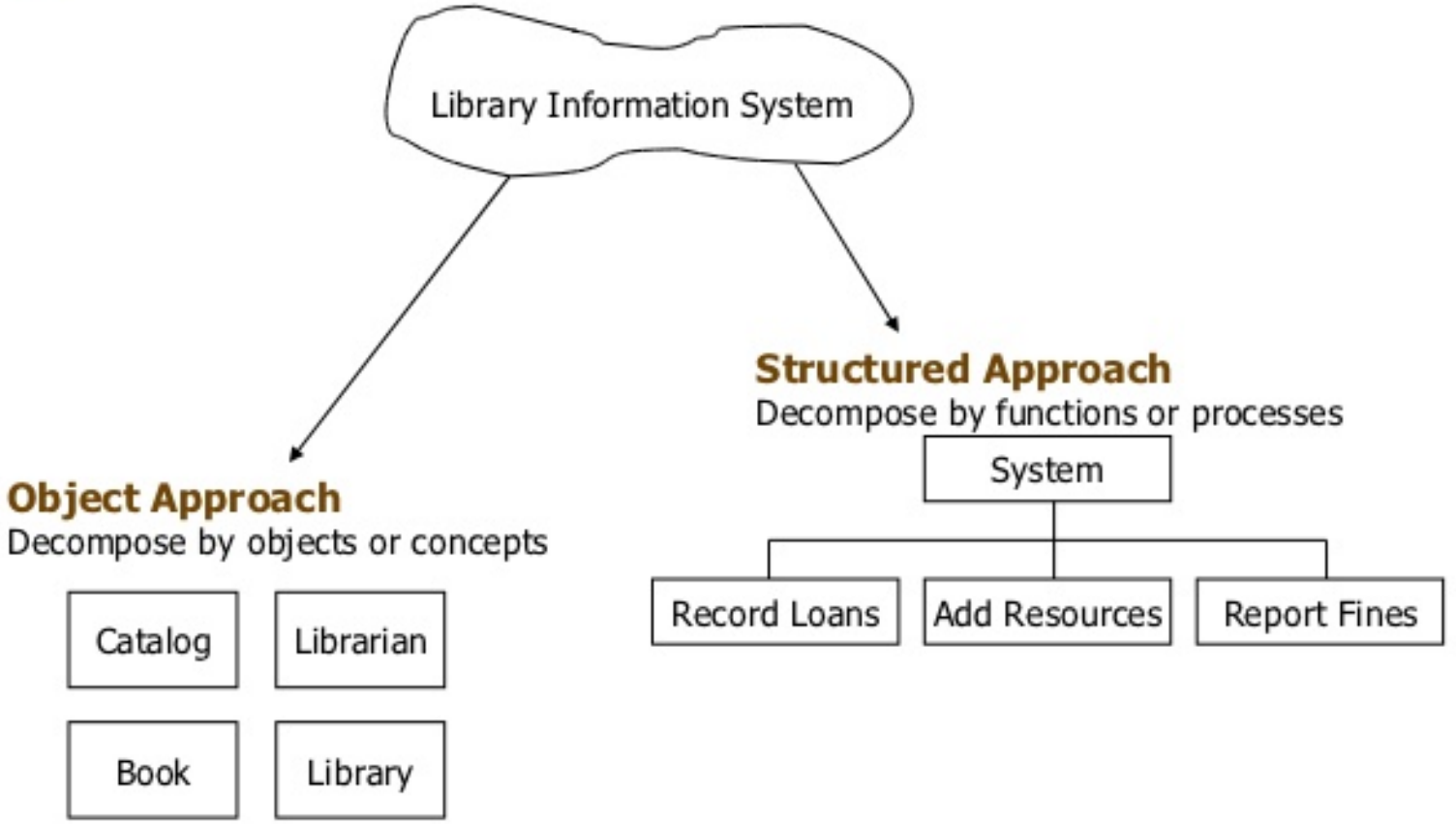
Structured



Object-oriented



Modularity: Structured vs. Object-oriented



Source: Craig Larman, *Applying UML and Patterns* (Prentice Hall, 1998), p. 14

Today

- **Software as a Complex Thing**
  - Dealing with Complexity
  - Functional Decomposition
  - Structured Programming

- **Object Oriented Paradigm**
  - Principles of Object Orientation
  - Classes and Objects
  - Sample Object Designs

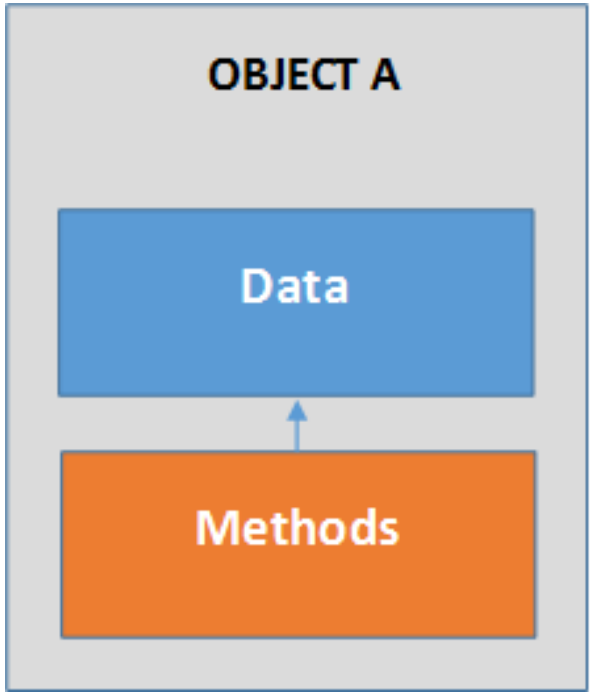
"Sir Isaac Newton secretly admitted to some friends: He understood how gravity behaved, but not how it worked!"

LILY TOMLIN

The Search for Signs of Intelligent Life in the Universe

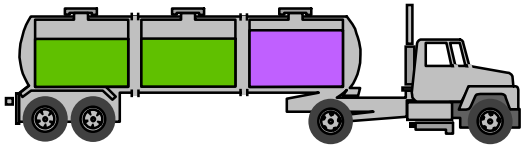
Object-Oriented Paradigm

- Centered on the concept of the object
- Object
  - Is data with methods
    - Data (**attributes**) can be simple things like number or character strings, or they can be other objects.
  - Defines things that are responsible for themselves
    - Data to know what state the object is in.
    - **Method** (code) to function properly.



What is an Object?

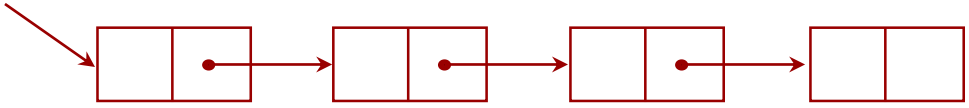
- Informally, an object represents an entity which is either physical, conceptual or software.
  - Physical entity
  - Conceptual entity
  - Software entity



Truck

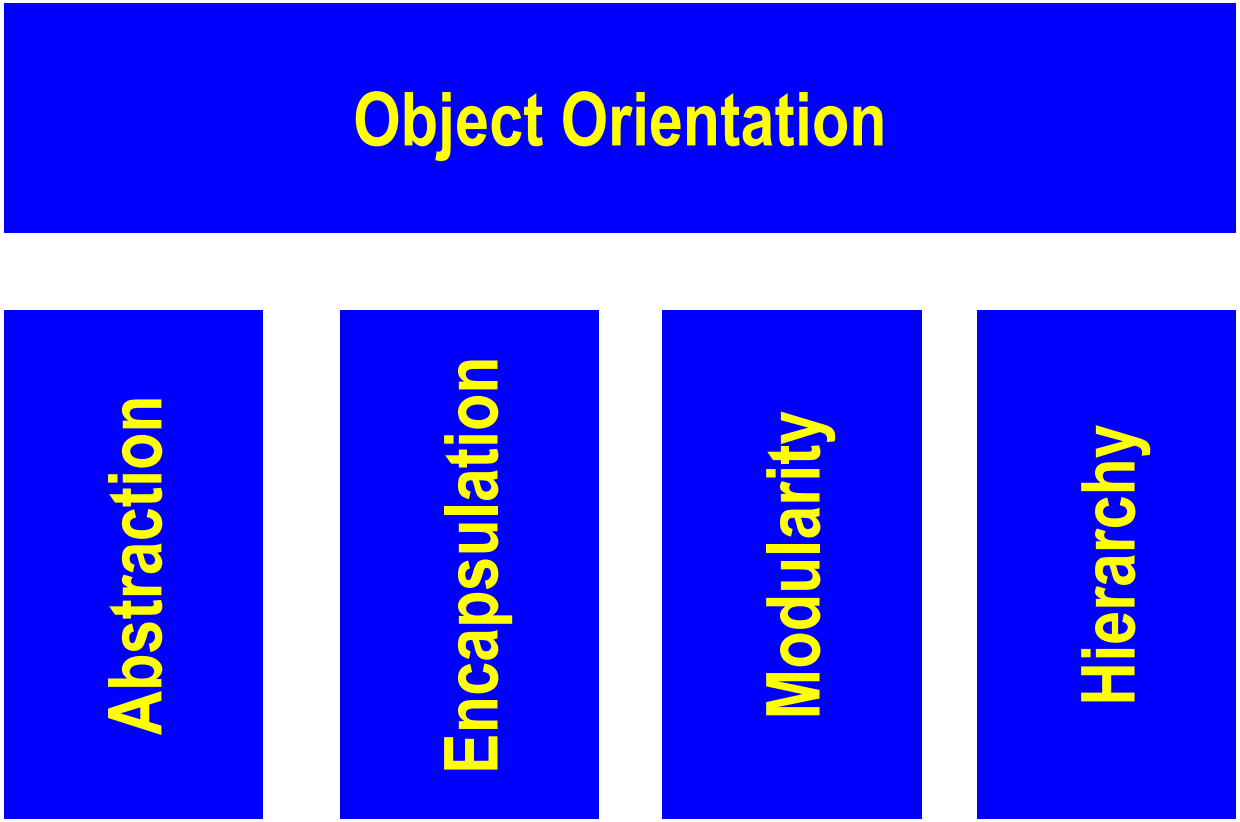
Chemical Process

Bank Account



Linked List

Basic Principles of Object Orientation



What is Abstraction?

Abstraction is one of the fundamental ways that we as humans cope with complexity.

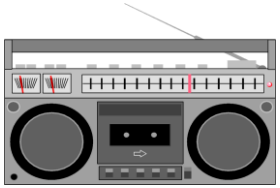
Salesperson

Customer

Not saying

which salesperson

just a salesperson in general!



Product

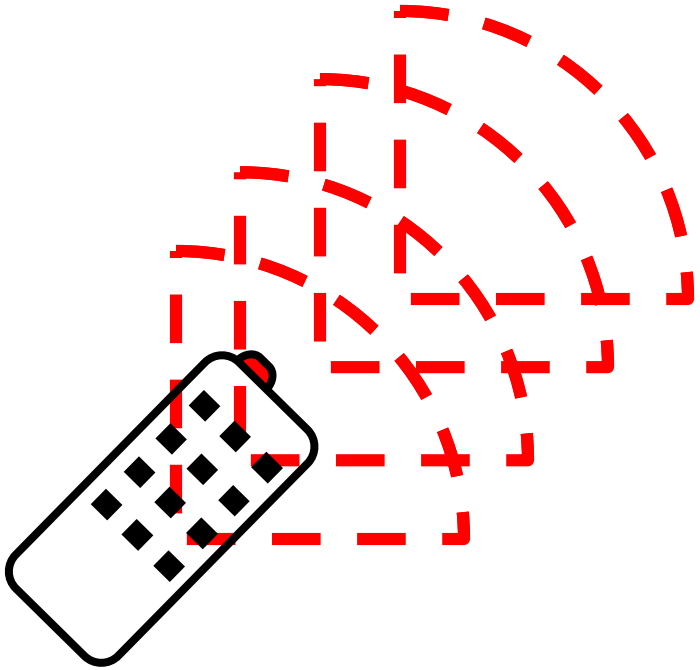
Dahl, Dijkstra, and Hoare suggest that "abstraction arises from a **recognition of similarities** between certain objects, situations, or processes in the real world, and the decision to ***concentrate upon these similarities and to ignore for the time being the differences***".

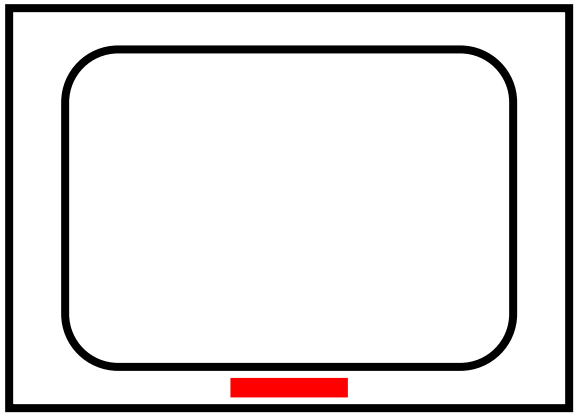
What is Abstraction?

Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

What is Encapsulation?

- Hide implementation from clients
  - Clients depend on interface

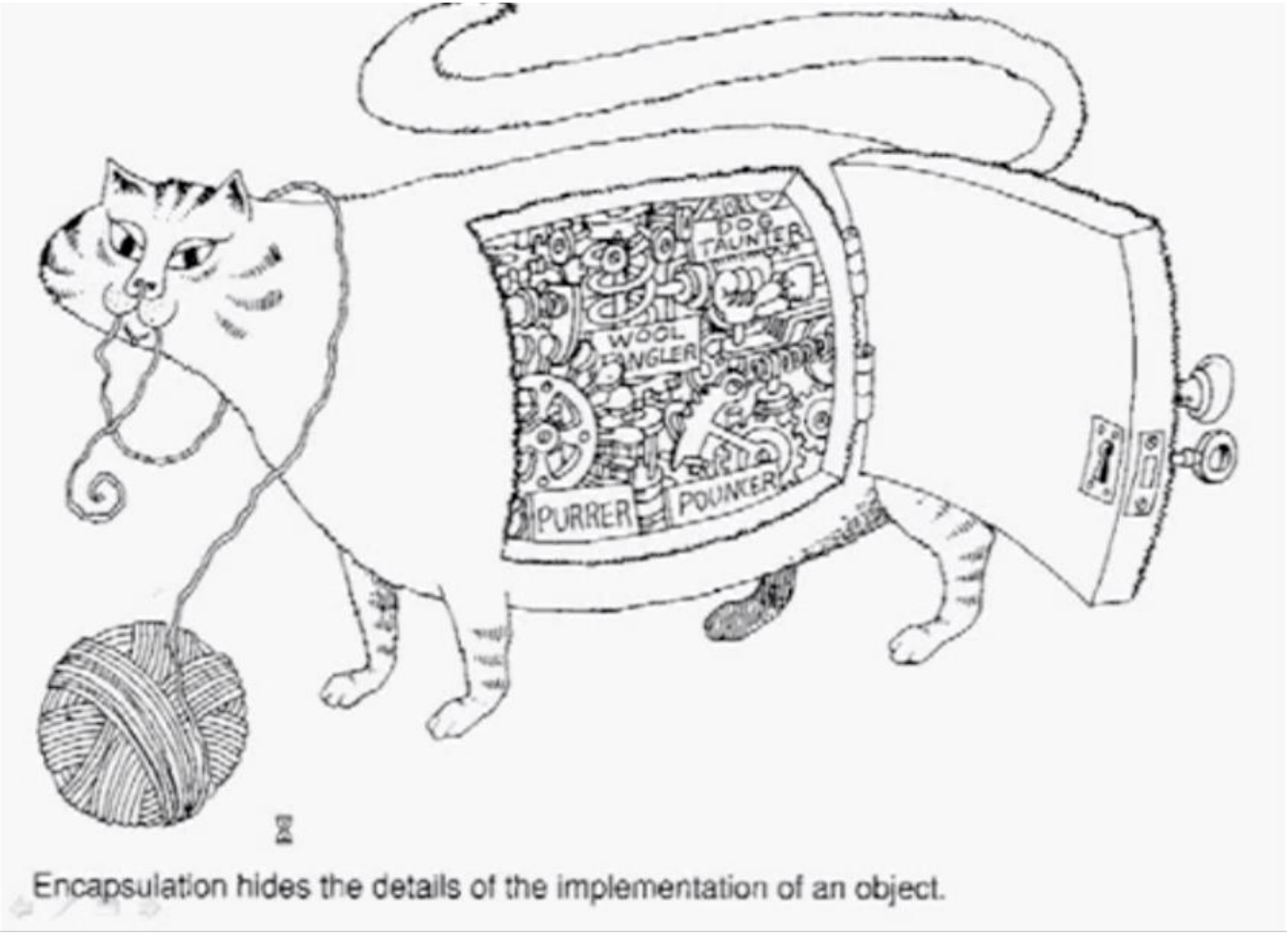




Information Hiding:
How does an object encapsulate?
What does it encapsulate?

Abstraction and encapsulation are complementary concepts: Abstraction focuses on the observable behavior of an object, whereas encapsulation focuses on the implementation that gives rise to this behavior.

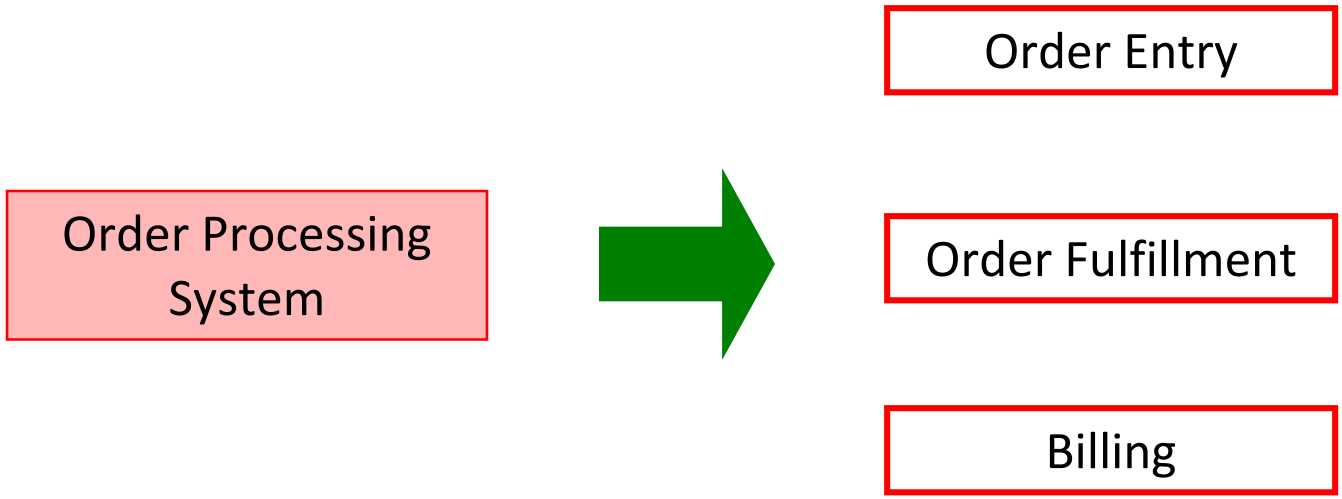
What is Encapsulation?



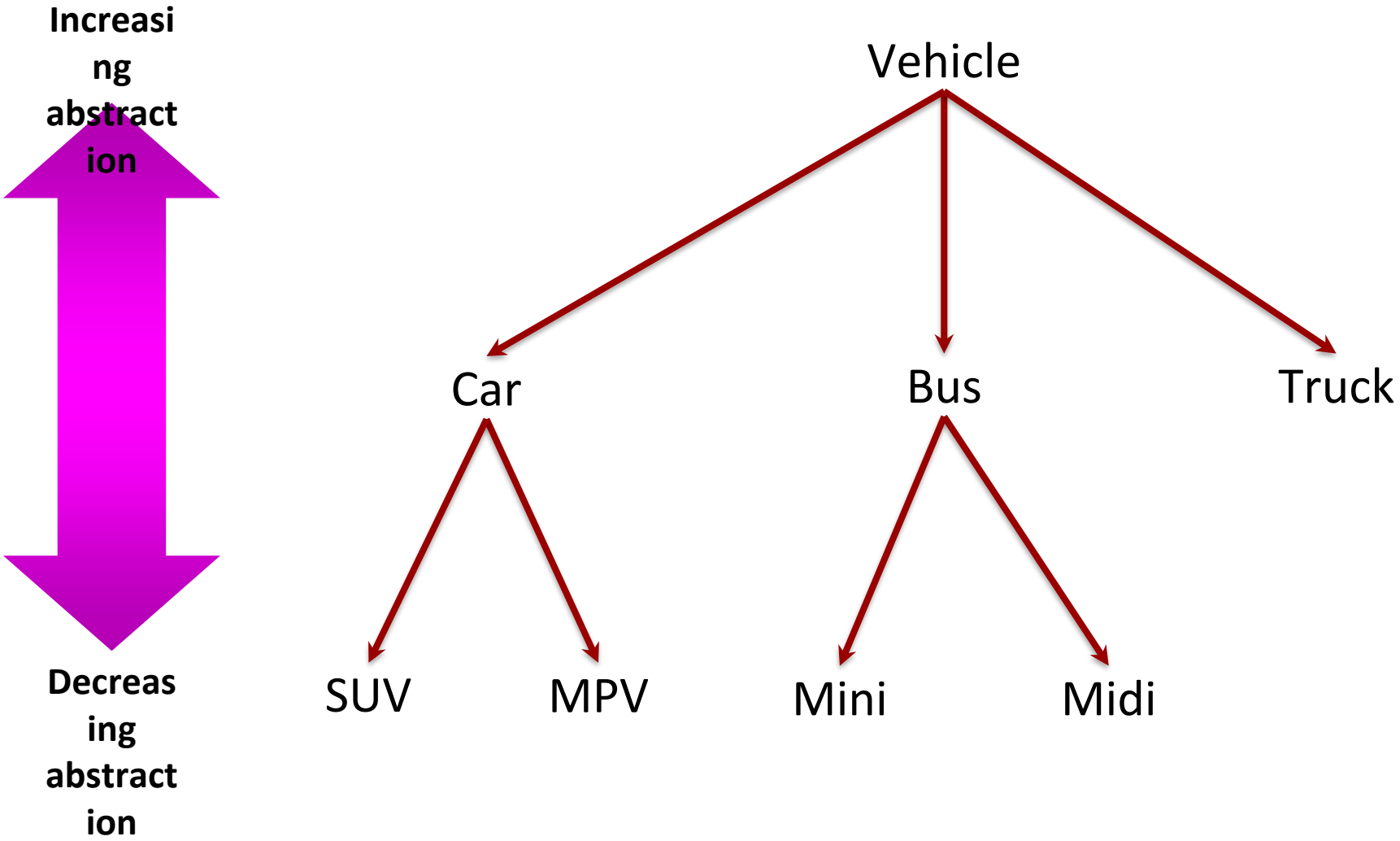
Encapsulation hides the details of the implementation of an object.

What is Modularity?

- The breaking up of something complex into manageable pieces.



What is Hierarchy?



Elements at the same level of the hierarchy should be at the same level of abstraction.

What is Really an Object?

- Formally, an object is a concept, abstraction, or thing with sharp boundaries and meaning for an application.

- An object is something that has:
  - State (property, attribute)
  - Behavior (operation, method)
  - Identity

Representing Objects

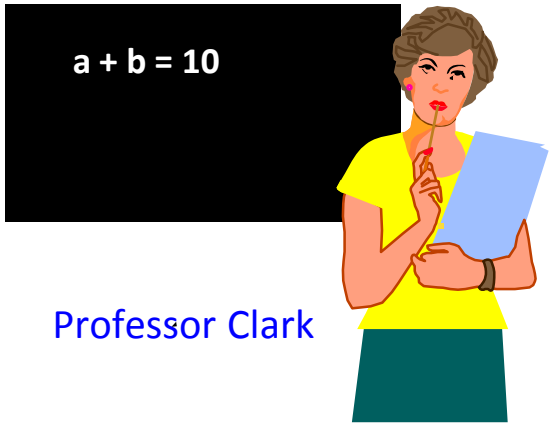
- An object is represented as rectangles with underlined names.

: Professor

Class Name Only

ProfessorClark

Object Name Only



Professor Clark

ProfessorClark : Professor

Class and Object Name

What is a Class?

- A class is a description of a group of objects with common properties (attributes), behavior (operations), relationships, and semantics
  - An object is an instance of a class
- A class is an abstraction in that it:
  - Emphasizes relevant characteristics
  - Suppresses other characteristics

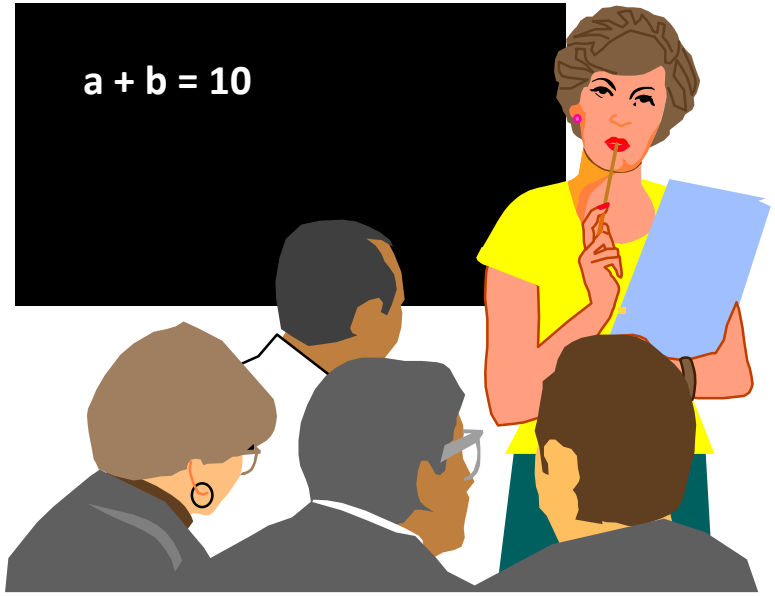
Example Class

Class

Course

Properties

Name
Location
Days offered
Credit hours
Start time
End time

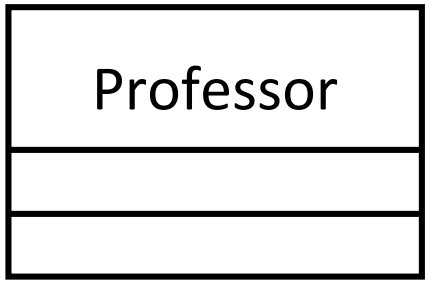


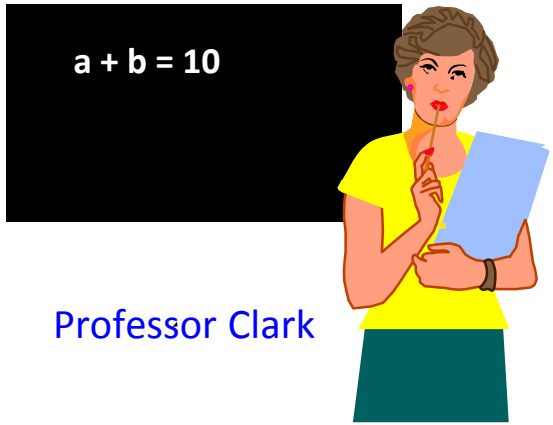
Behavior

Add a student
Delete a student
Get course roster
Determine if it is full

Representing Classes

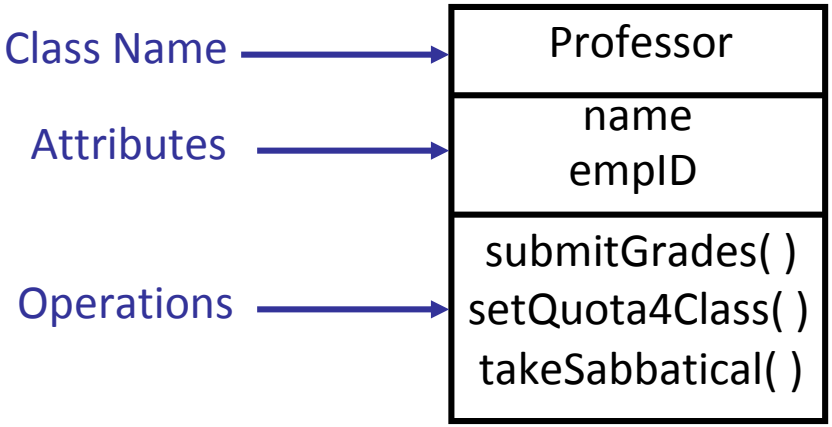
- A class is represented using a compartmented rectangle



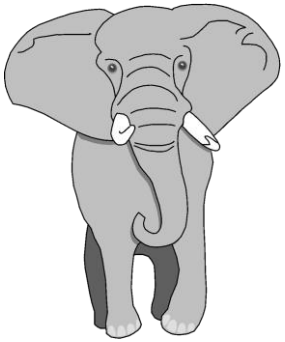
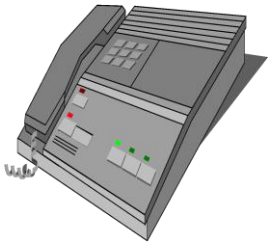
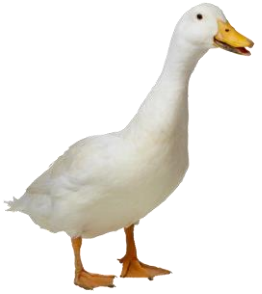


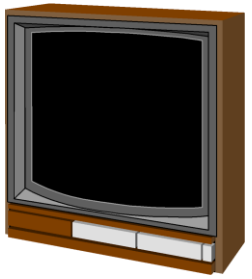
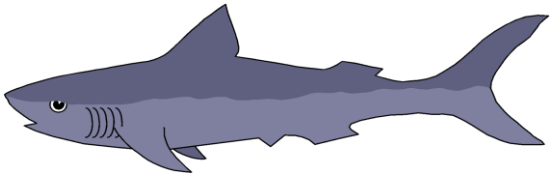
Class Compartments

- A class is comprised of three sections
  - The first section contains the **class name**
  - The second section shows the **structure** (attributes)
  - The third section shows the **behavior** (operations)



How Many Classes do you See?


Relationship between Classes and Objects

- A class is an abstract definition of an object
  - It defines the structure and behavior of each object in the class
  - It serves as a template for creating objects
- Objects are grouped into classes

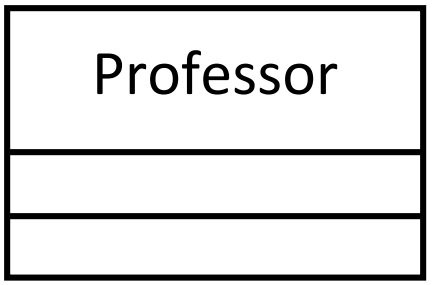
Professor Smith

Objects

Professor Jones

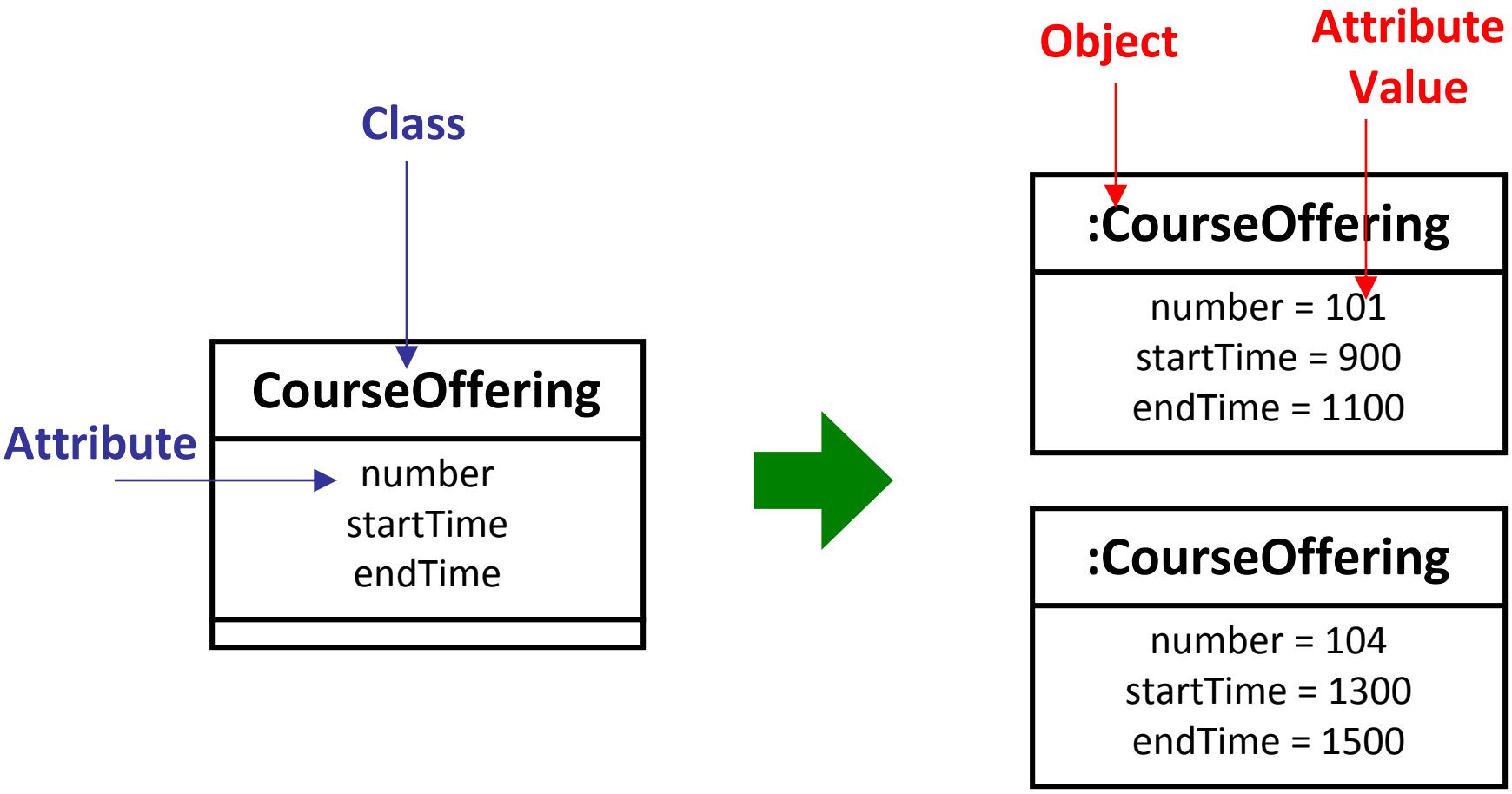
Professor Mellon

Class



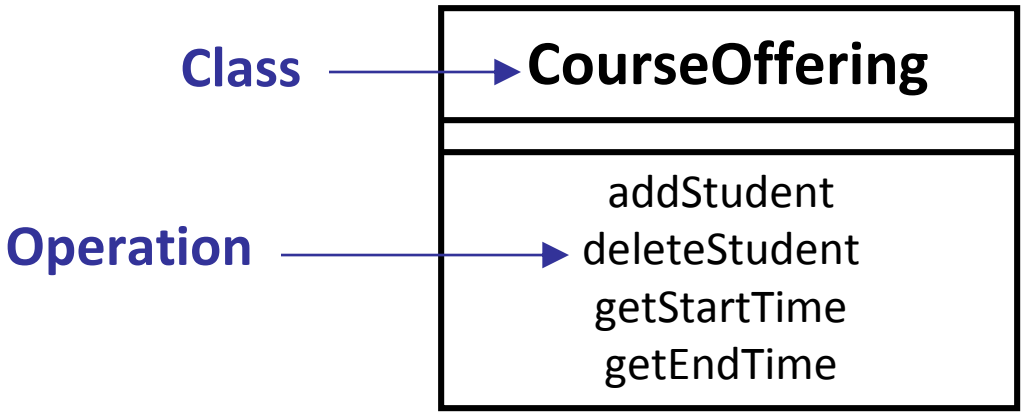
State of an Object (property or attribute)

- The state of an object encompasses all of the (usually static) properties of the object plus the current (usually dynamic) values of each of these properties.



Behavior of an Object (operation or method)

- Behavior is how an object acts and reacts, in terms of its state changes and message passing.



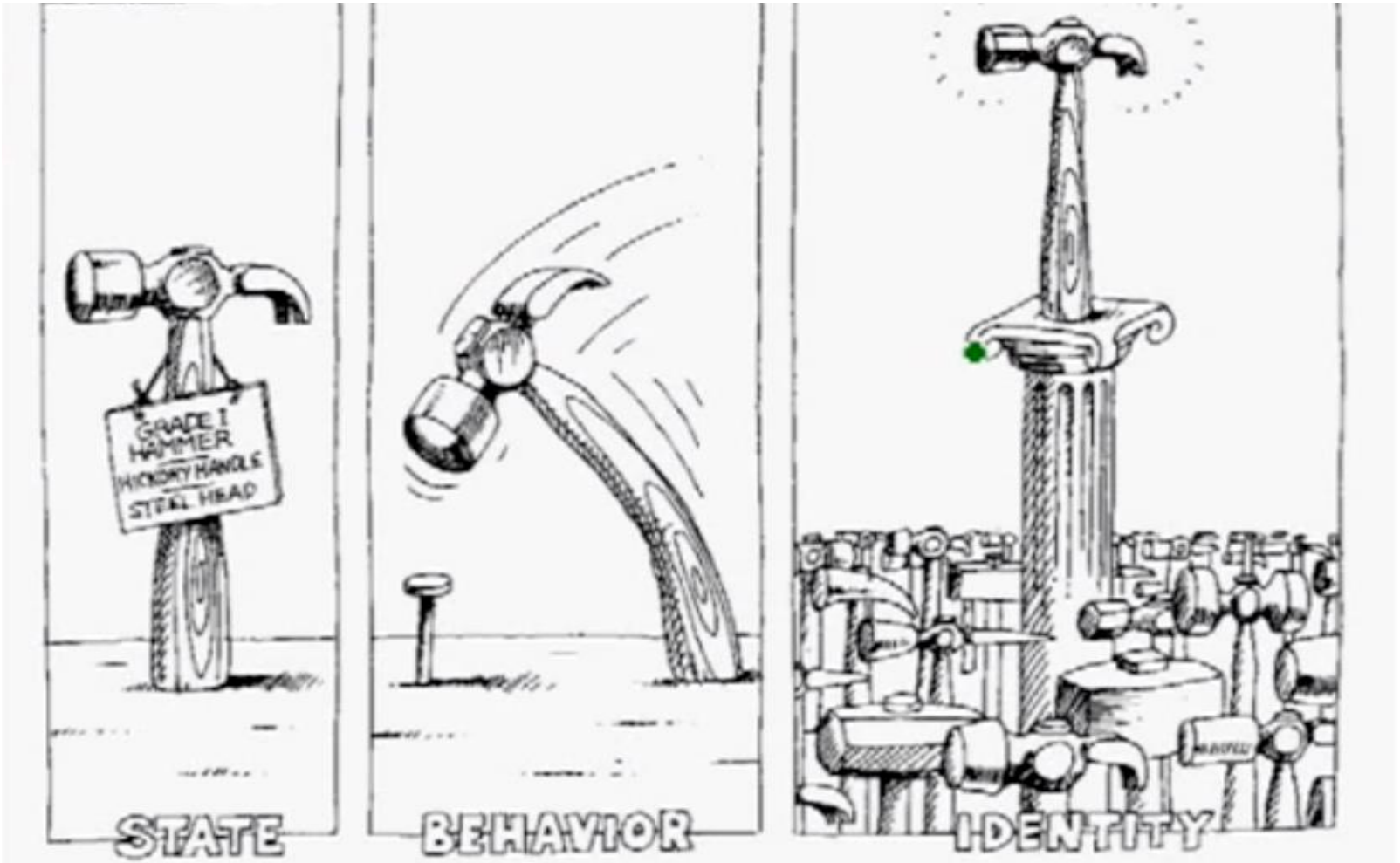
Identity of an Object

- Each object has a unique identity, even if the state is identical to that of another object.

Professor "J Clark" teaches Biology

Professor "J Clark" teaches Biology

A Class



An object has state, exhibits some well-defined behavior, and has a unique identity.

Sample Class: Automobile

- Attributes
  - manufacturer's name
  - model name
  - year made
  - color
  - number of doors
  - size of engine

- Methods
  - Define attributes (specify manufacturer's name, model, year, etc.)
  - Change a data item (color, engine, etc.)
  - Display data items
  - Calculate cost

Sample Class: Circle

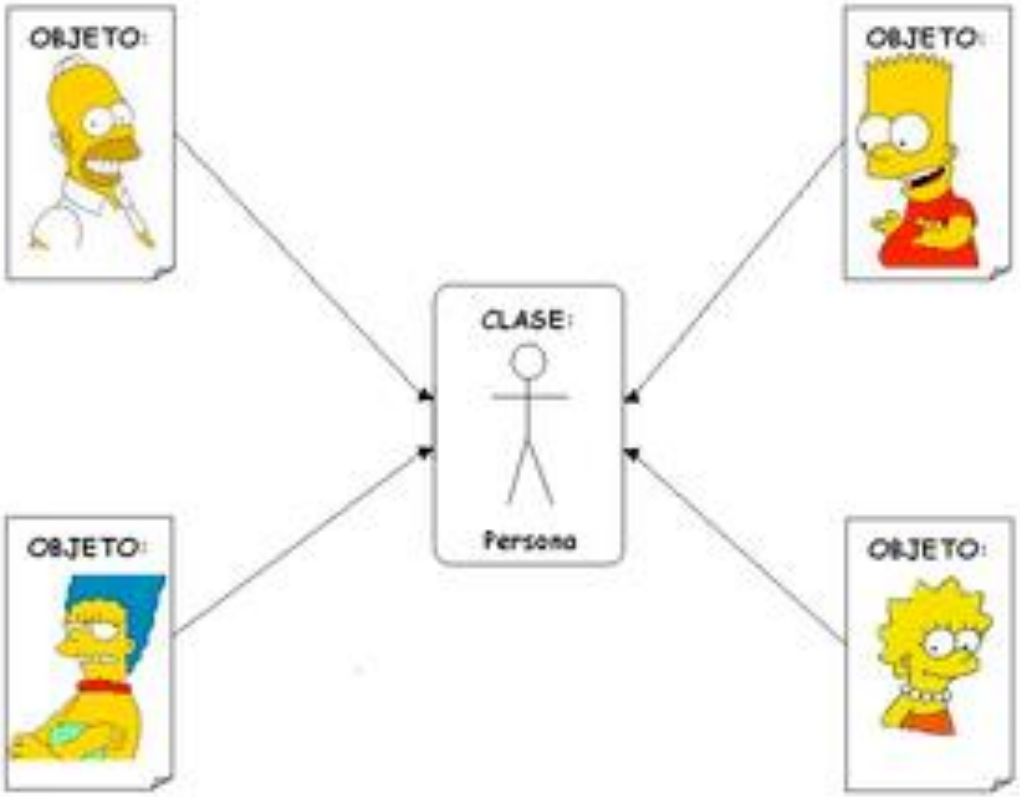
- Attributes
  - Radius
  - Center Coordinates
    - X and Y values

- Methods
  - Define attributes (radius and center coordinates)
  - Find area of the circle
  - Find circumference of the circle

Sample Class: Baby

- Attributes
  - Name
  - Gender
  - Weight
  - Decibel
  - # poops so far
- Methods
  - Get or Set specified attribute value
  - Poop

Sample Class: Person



Summary

- So far, we covered basics of objects and object oriented paradigm.
  - We tried to think in terms of objects.
- From now on, we should be seeing objects everywhere ☺
  - Or, we should be realizing that we were seeing objects everywhere already.
  - This is actually something you do naturally. Why not do programming that way?
- We will continue next week with actually creating objects by using Java.

Acknowledgments

- The course material used to prepare this presentation is mostly taken/adopted from the list below:
  - Software Enginering (9th ed) by I.Sommerville, Pearson, 2011.
  - Object Oriented Analysis and Design with Applications, Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Conallen and Kelli A. Houston, Addison Wesley, 2007.
  - OOAD Using the UML - Introduction to Object Orientation, v 4.2, 1998-1999 Rational Software
  - Java - An Introduction to Problem Solving and Programming, Walter Savitch, Pearson, 2012.
  - Ku-Yaw Chang, Da-Yeh University.