Module 1:
Introduction to Object Oriented Programming

Dr. Tran Minh Triet

Acknowledgement

- This presentation reuses materials from:
  - Course CS202: Programming Systems
    Instructor: MSc. Karla Fant,
    Portland State University
  - Course CS202: Programming Systems
    Instructor: Dr. Dinh Ba Tien,
    University of Science, VNU-HCMC
  - Course DEV275: Essentials of Visual Modeling with UML 2.0
    IBM Software Group

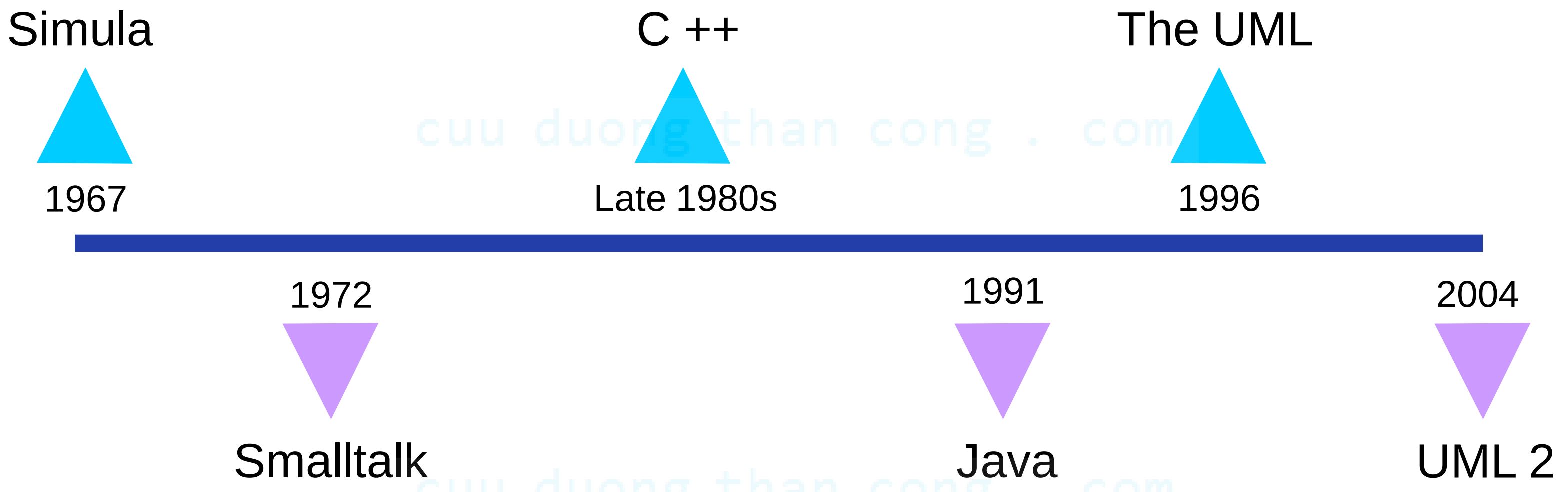
Outline

- Introduction
- Procedural vs. OO Programming
- Four principles of OO
- Other issues

Introduction…

The History of Object Technology

- Major object technology milestones



Object-oriented concepts

- Learning OO concepts is not accomplished by learning a specific development method or a set of tools.

- But, it is a way of thinking.

Object-oriented concepts (cont)

For examples:

- Many people are introduced to OO concepts via one of these development methods or tools.
  - Many C programmers were first introduced to object orientation by migrating directly to C++, before they were even remotely exposed to OO concepts.
  - Some software professionals were first introduced to object orientation by presentations that included object models using UML

Problems!!!

- Learning a programming language is an important step, but it is much more important to learn OO concepts first.
  - Developers who claim to be C++ programmers are simply C programmers using C++ compilers.
  - Learning UML before OO concepts is similar to learning how to read an electrical diagram without first knowing anything about electricity.

Even worse!!!

- A programmer can use just enough OO features to make a program incomprehensible to OO and non-OO programmers alike.

Practice

- Using the data structure struct, implement the following structures and functions:
  - struct **Point**: data structure for a 2-D point.
  - struct **Triangle**: contains the information of the 3 vertices
  - A function to calculate a distance between 2 points
  - Functions to calculate the perimeter and area of a triangle.

We already know everything about OOP???

OO concepts

It is very important that while you're on the road to OO development, you first learn the fundamental **OO concepts**.

Procedural vs. OO Programming

Procedural vs. OO Programming

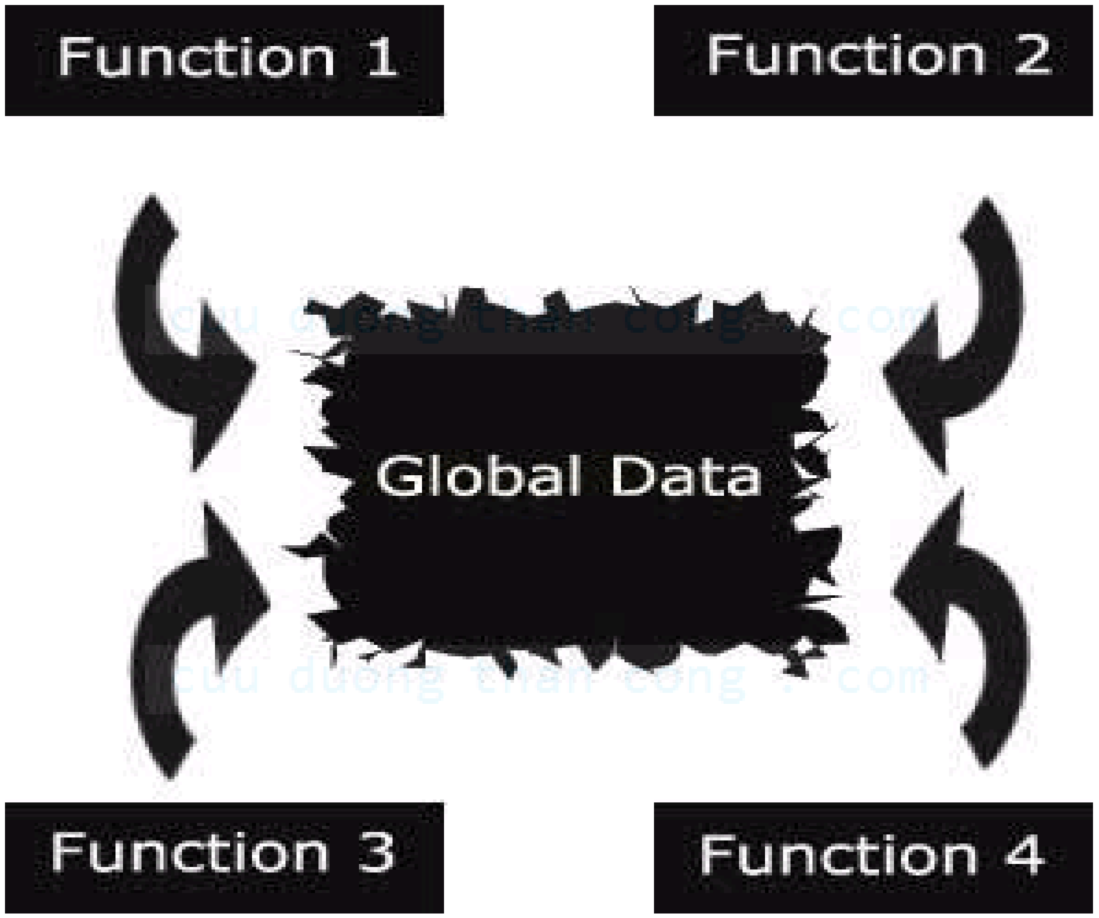
An object is an entity
that contains both data and behaviours

- In procedural programming:
  - Code is placed into totally distinct functions or procedures.
  - Data is placed into separate structures, and is manipulated by these functions or procedures.

Procedural vs. OO Programming (cont)

- In OO programming: the attributes and behaviours are contained within a single object
- In procedural programming: the attributes and behaviours are normally separated.

Why do we change from procedural to OO programming?



Why do we change from procedural to OO programming?

- In procedural programming:
  - Data is separated from the procedures.
  - Sometimes it is global → easy to modify data that is outside your scope

  → This means that access to data is uncontrolled and unpredictable.
  - Having no control over the data → testing and debugging are much more difficult.

Why do we change from procedural to OO programming?

- Objects solve these problems by combining data and behaviours into a complete package.

- In a proper OO design: there is no global data (we expect ;-)

Objects (again!)

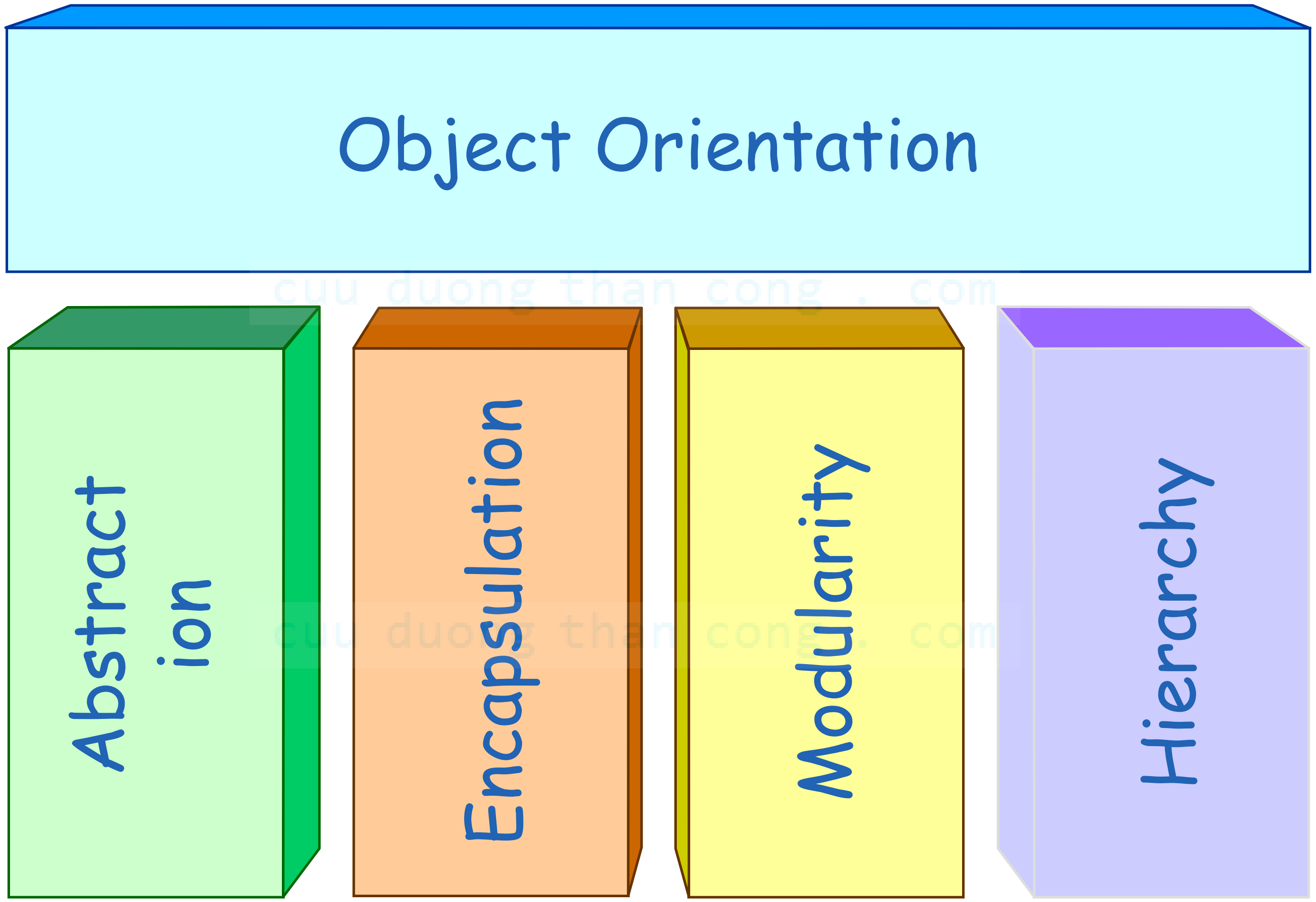
- Objects do contain:
  - Integers, and strings… → attributes.
  - Methods (i.e. functions) → behaviours.
- In an object, methods are used to operate on the data.

You can control access to

members of an object (both attributes and methods).

Four principles of OO

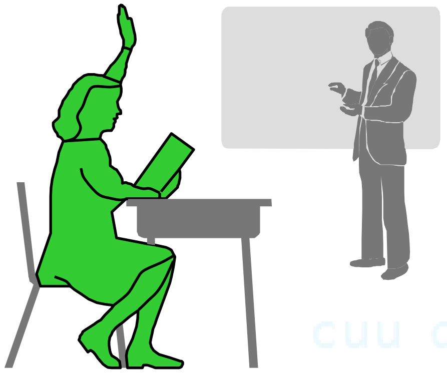
Basic Principles of Object Orientation

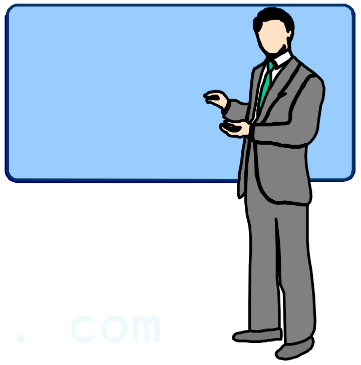


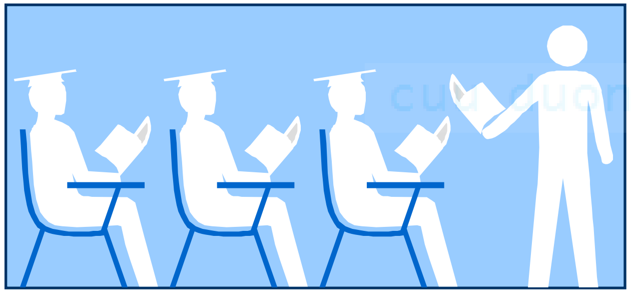
What Is Abstraction?

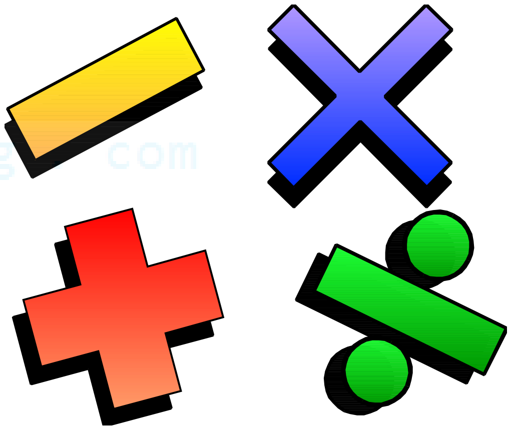
- The essential characteristics of an entity that distinguishes it from all other kinds of entities.
- Defines a boundary relative to the perspective of the viewer.
- Is not a concrete manifestation, denotes the ideal essence of something.

Example: Abstraction


Student

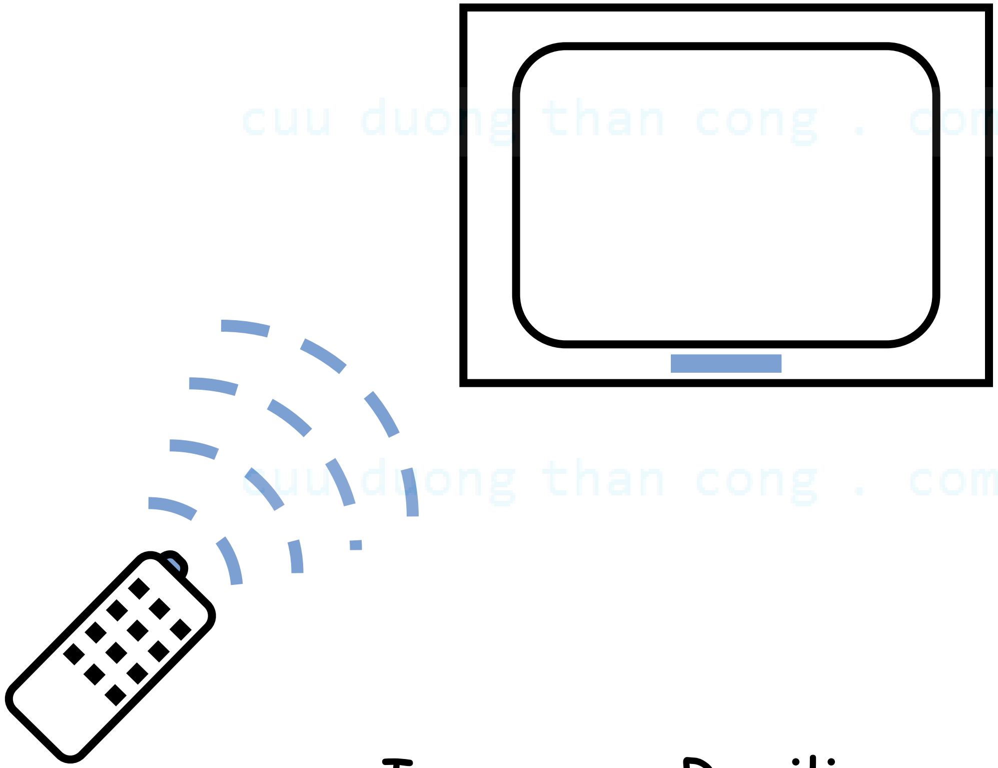

Professor


Course Offering (9:00 a.m., Monday-Wednesday-Friday)


Course (e.g. Algebra)

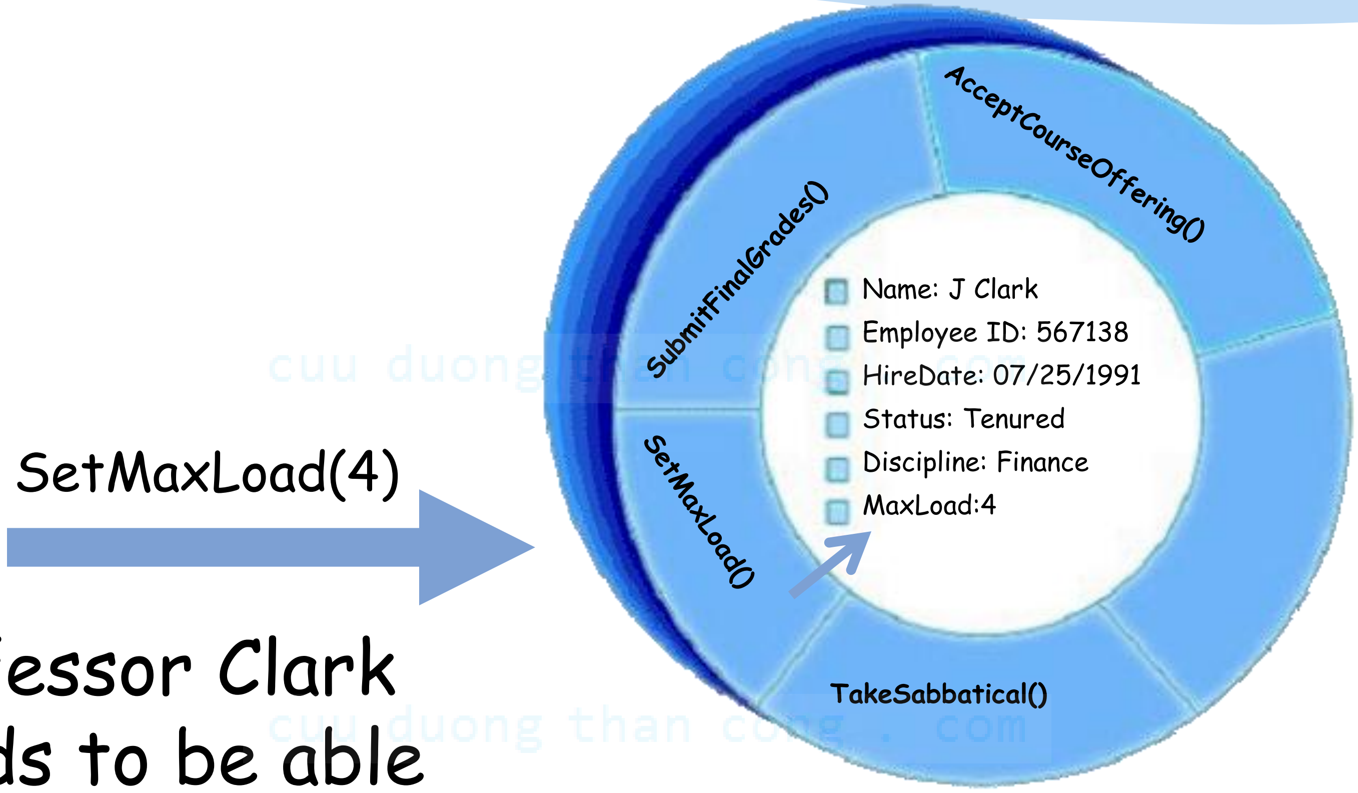
What Is Encapsulation?

- Hides implementation from clients.
  - Clients depend on interface.



Improves Resiliency

Encapsulation Illustrated

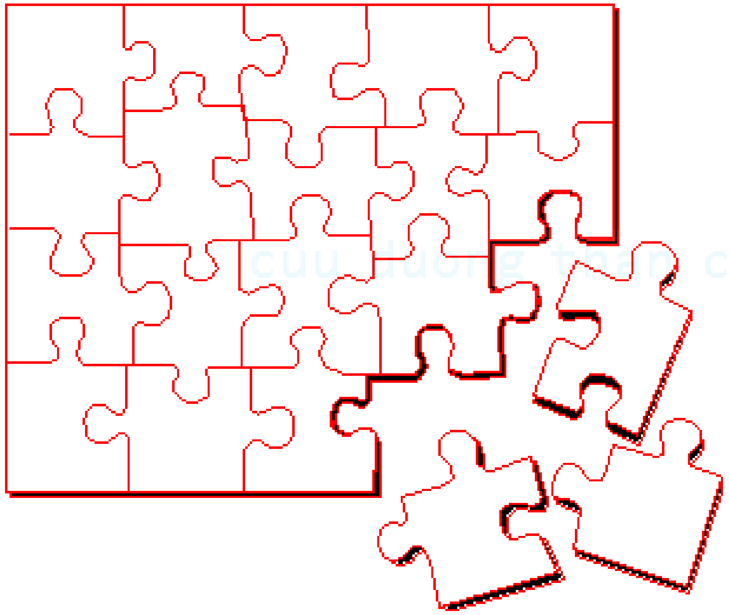


Professor Clark

- Professor Clark needs to be able to teach four classes in the next semester.

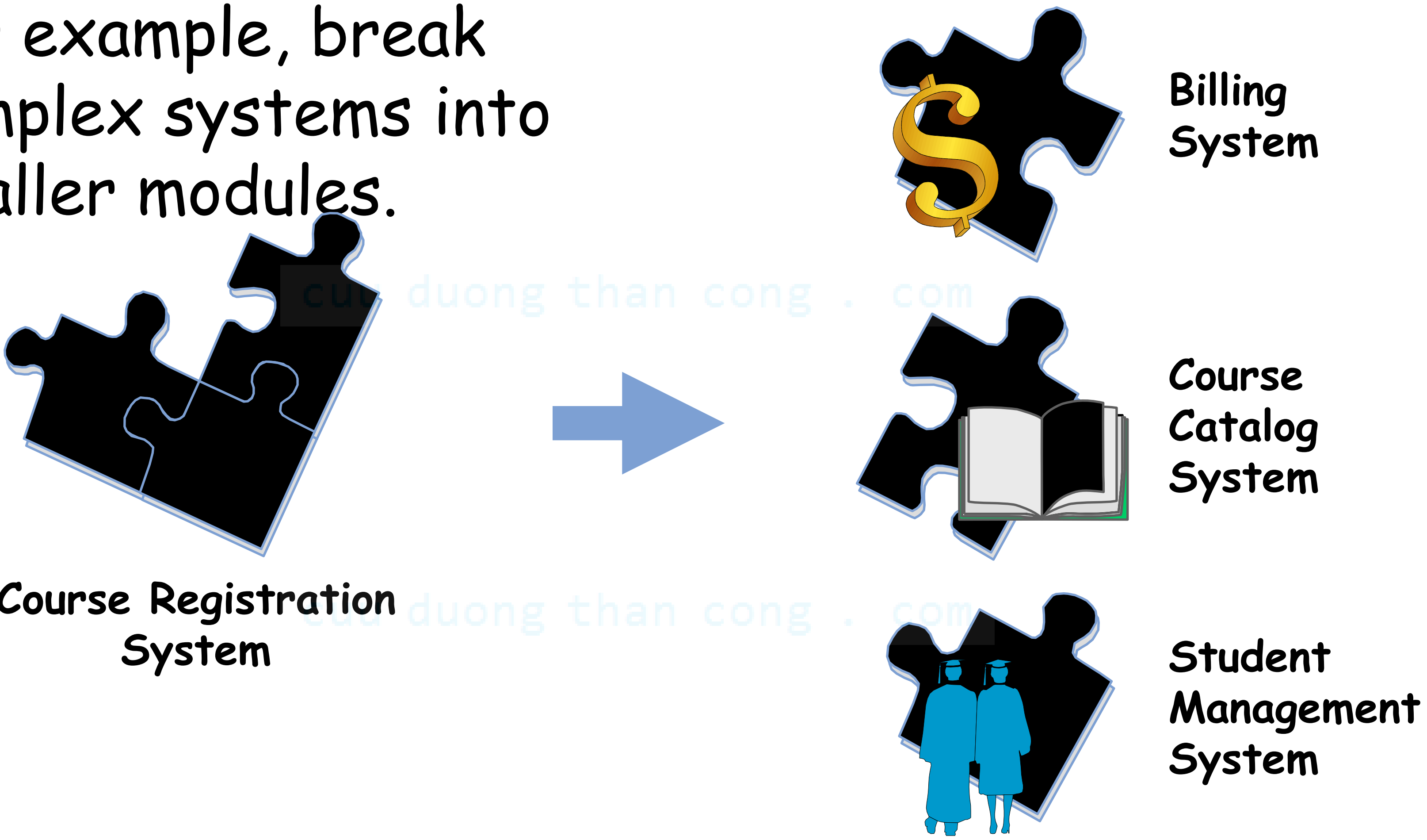
What Is Modularity?

- Breaks up something complex into manageable pieces.
- Helps people understand complex systems.

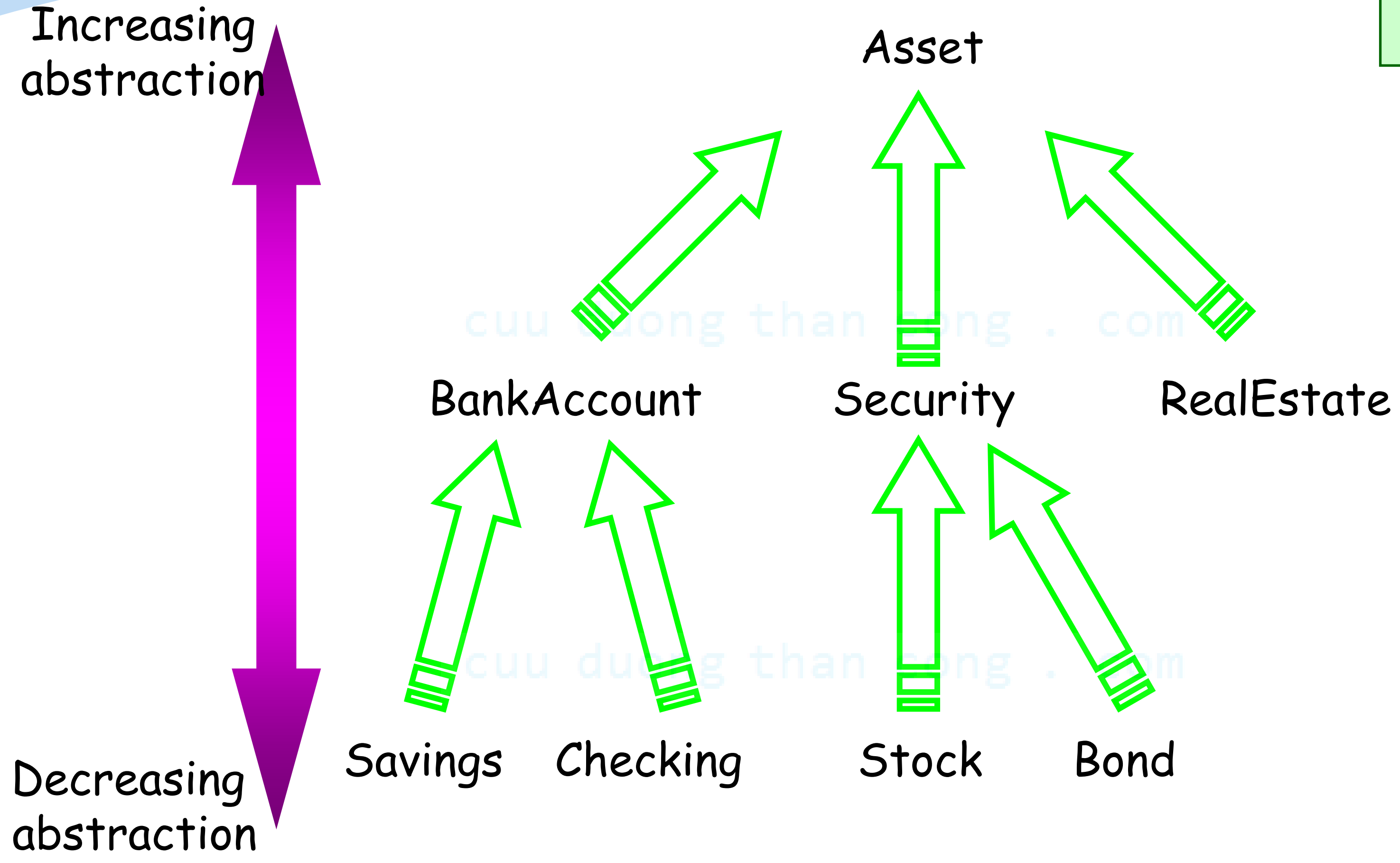


Example: Modularity

- For example, break complex systems into smaller modules.



What Is Hierarchy?



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

Other issues…

Software Design

- Reusability
  - Portable and independent components can be reused in many systems
- Extensibility
  - Support external plug-ins
- Flexibility
  - Change will be easy when new data/features added
  - Modifications are less likely to break the system
  - Localize effect of changes

Create a system: designing process

- Divide a system in terms of components
- Divide components in terms of sub-components
- Abstraction
  - Hides details of components that are irrelevant to the current design phase
- Component identification is top-down
  - Decompose system into smaller, simple components
- Integration is bottom-up
  - Combining small components
- Design is applied using a paradigm: procedural, modular, objectoriented

Abstraction

- Procedural design
  - Define set of functions to accomplish task
  - Pass information from function to function
- Modules (modular design)
  - Define modules, where each has data and procedures
  - Each module has a public and a private section
  - Works as a scoping mechanism
- Classes/Objects (object-oriented design)
  - Abstract Data Types
  - Divide project in set of cooperating classes
  - Each class has a very specific functionality
  - Classes can be used to create multiple instances of objects