Introduction to Computational Thinking

Encapsulation

Qiao Xiang, Qingyu Song

https://sngroup.org.cn/courses/ct-xmuf25/index.shtml

12/6/2025

This deck of slides are heavily based on cs112 at Yale University and cs101 at UCAS, respectively, by courtesy of Dr. Y. Richard Yang and Dr. Zhiwei Xu.

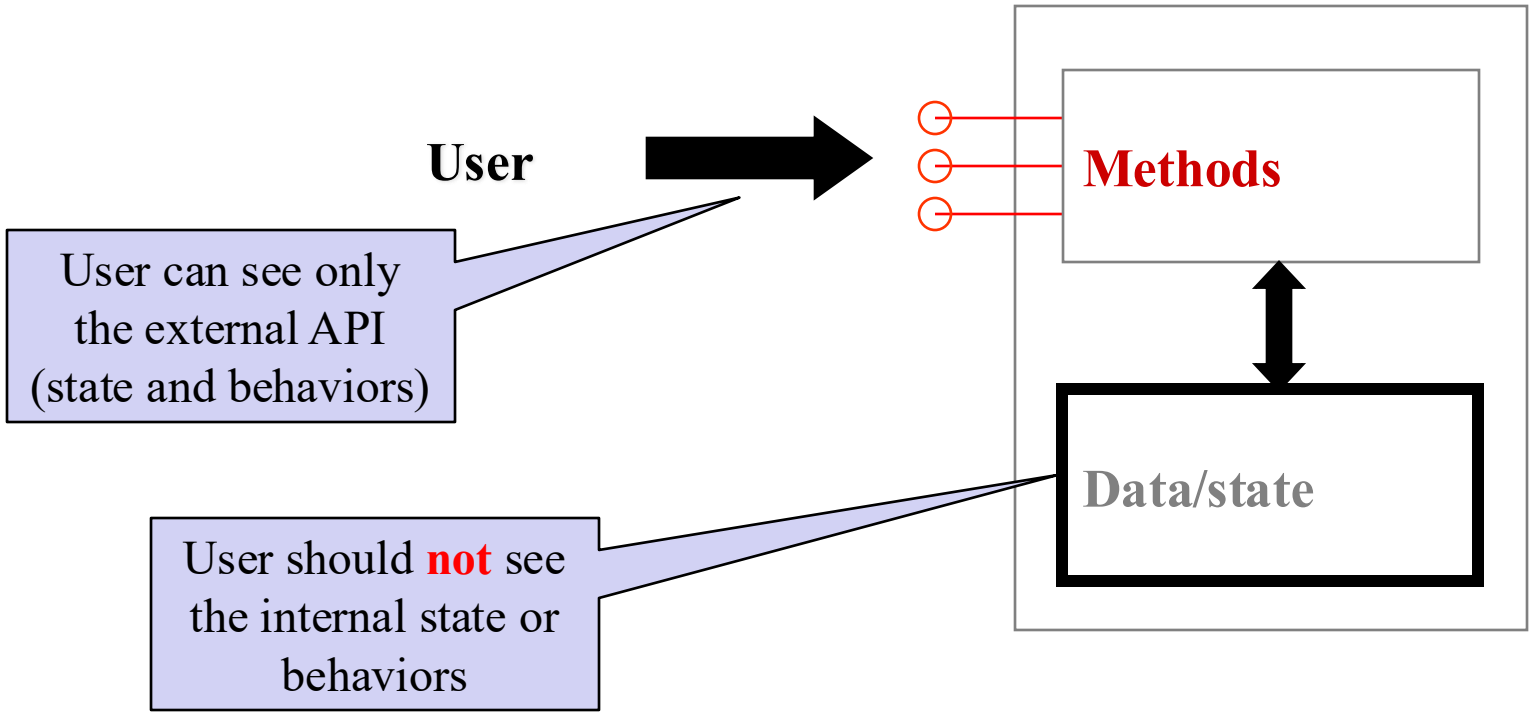
Outline

- The encapsulation(封装) principle

Two Views of an Object

- You can take one of two views of an object:
  - external (API) - the interaction of the object with its users
  - internal (implementation) - the structure of its data, the algorithms used by its methods

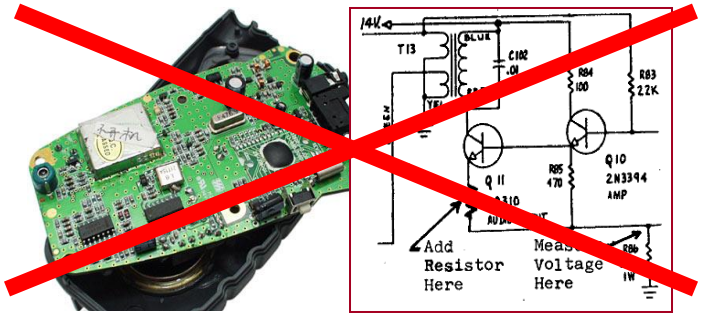
The Encapsulation Principle



Encapsulation Analogy

Client

API



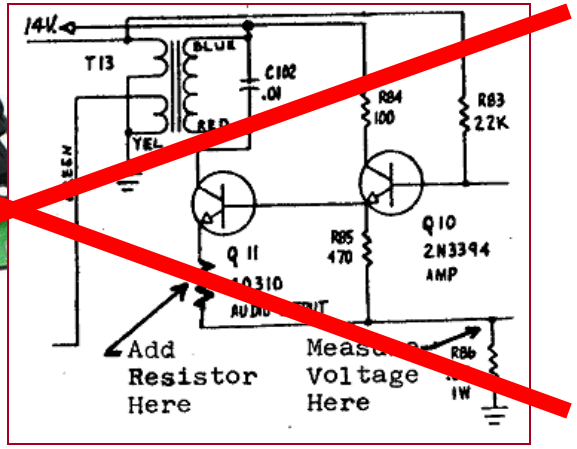
Implementation

client needs to know
how to use API

implementation needs to know
what API to implement

Encapsulation Analogy

- As a user, you don't understand the inner details of iPhone, and you don't need to.
- Apple does not want to commit to any internal details so that Apple can continuously update the internal



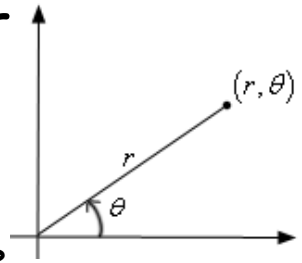
Examples

- What are some risks of allowing others to view/access the internal state of a class?
  - The Coin class
  - The Ball class
  - The Point class
  - The BankAccount class

Why Encapsulating Data

- Consistency: so that we make it impossible for others to "reach in" and directly alter another object's state
  - Protect object from unwanted access
    - Example: Can't fraudulently increase an `BankAccount` balance.
  - Maintain state **invariants**
    - Example: Only allow `BankAccount`s with non-negative balance.
    - Example: Only allow `Date`s with a month from 1-12.

- Flexibility: so that you can change the state representation later without worrying about breaking others' code
  - Example: `Point` could be rewritten in polar coordinates (r, θ) so long you provide translation in the interface, clients will not see difference.



Accomplish Encapsulation: Access Modifiers

- In Java, we accomplish encapsulation through the appropriate use of *access modifiers*(修饰符)

- An access modifier is a Java reserved word that specifies the accessibility of a method, data field, or class
  - we will discuss two access modifiers: `public`, `private`
  - we will discuss the other modifier (`protected`) later

The `public` and `private` Access Modifiers

- access modifiers enforce encapsulation
  - **`public`** members (data and methods): can be accessed from **anywhere**
  - **`private`** members: can be accessed from a method defined in the **same class**
  - Members without an access modifier: default **private** accessibility, i.e., accessible in the same package; otherwise, not accessible.

Using Access Modifiers to Implement Encapsulation: Methods

- Only service methods should be made `public`
- Support or helper methods created simply to assist service methods should be declared `private`

The Effects of Public and Private Accessibility

	public	private
variables	violate Encapsulation Use Caution	enforce encapsulation
methods	provide services to clients	support other methods in the class