OBJECT-ORIENTED PROGRAMMING

Lecture 2: Object-oriented Programming (OOP) Concepts

Object-oriented Programming (OOP): A Closer Look

- Motivations of OOP
  - **Model real things** by objects.
  - **Code reuse**: “Don’t reinvent the wheel.”
  - **Simplify management of software**.
  - **Make software have predictable behavior.**

https://www.slideshare.net/Iqrakhalil2/object-oriented-programming-129281522

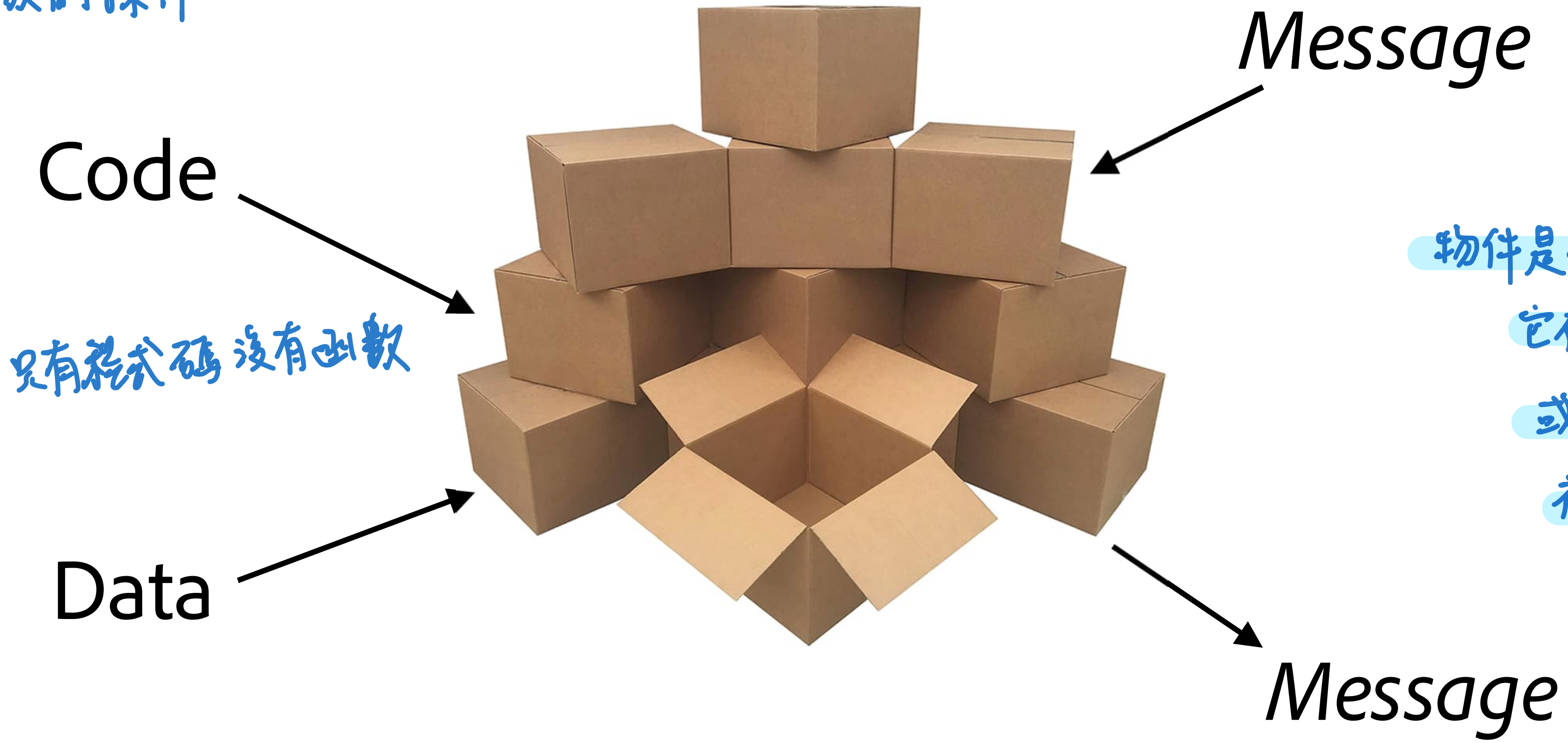
Object Are Black Boxes

物件

耦合.(coupling)

POD 類別

物件包含 { 物件本身的資料
對物件做的操作



Code

Data

Message

Message

只有程式碼 沒有函數

物件是透明的盒子.
它存在,但你看不到
或是黑盒子,
裡面有些資料
但你看不到.

https://web.csulb.edu/~mopkins/cecs277/index.shtml尢https://www.amazon.com/UBOXES-Moving-Inches-Bundle-BOXBUNDLAR12/dp/B007PBKMKK

Objects Are Building Blocks

https://www.mirrormedia.mg/story/20170414game_minecraft/

Objects Have Types

- *"Lucky is a cat."*

  案例、實例　　　　物件
  - Lucky is an **instance** or **object** of a cat.

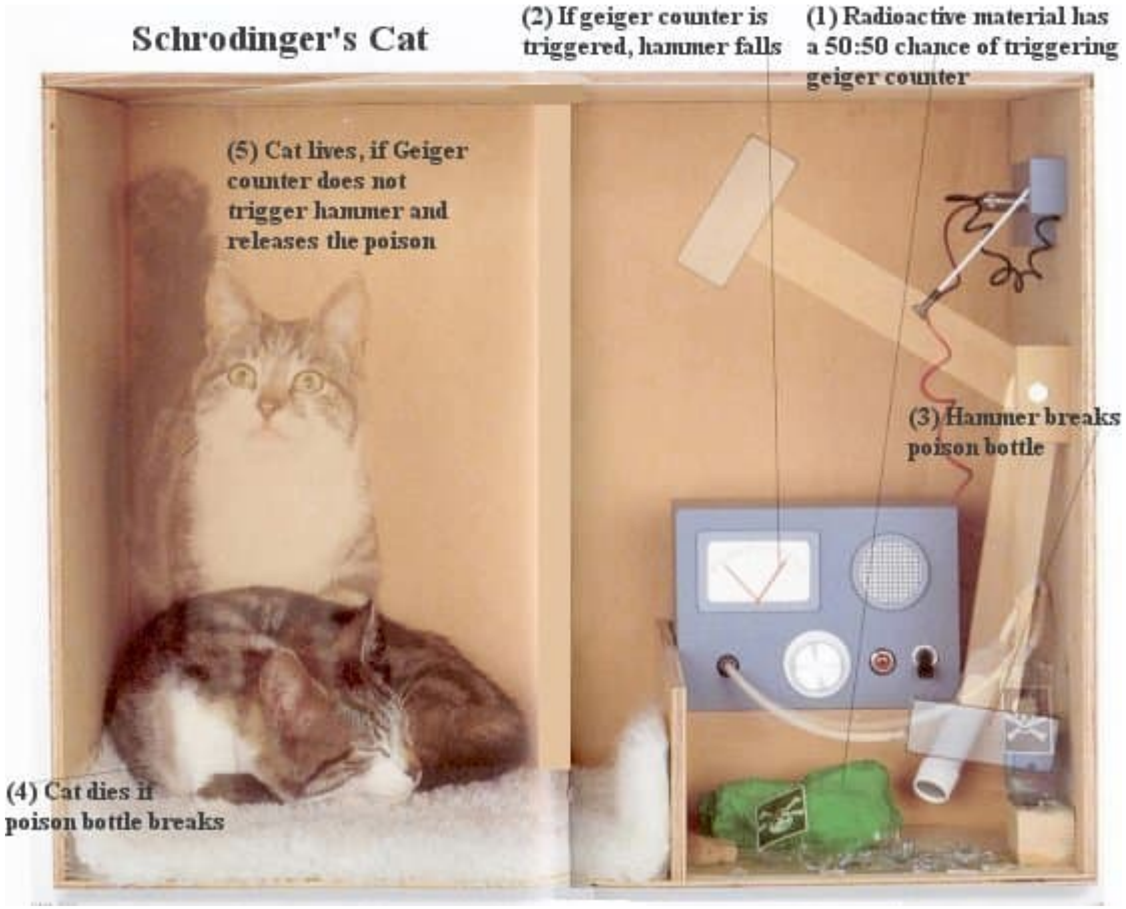
  型別　　類別
  - Cat is the **type** or **class** of Lucky.
  - "Lucky" is the **name** or **identifier** of the cat's instance.

  識別子

https://timesofindia.indiatimes.com/life-style/relationships/pets/5-things-that-scare-and-stress-your-cat/articleshow/67586673.cms

States of Objects

- An object can be under different **states**…


Schrodinger's Cat
(2) If geiger counter is triggered, hammer falls
(1) Radioactive material has a 50:50 chance of triggering geiger counter
(5) Cat lives, if Geiger counter does not trigger hammer and releases the poison
(3) Hammer breaks poison bottle
(4) Cat dies if poison bottle breaks

https://case.ntu.edu.tw/blog/?p=17466

Behavior of Objects

物件可以有狀態

- …and can take different actions and exhibit **behavior**.

I survived!

Take that, Schrodinger ☺!

https://www.dcard.tw/f/whysoserious/p/227677725

Messages

- 物件之間要溝通要透過訊息，不能互相存取資料

- **Principle**
  - Objects talk to each other by sending **messages** to each other.
  - Messages are passed to an object by calling some **method** of the object, during which some **parameters** are passed to the object.

物件的函數

→ 方法：專屬某類別的函數.

- **Objective**

把範圍縮小

  - **Information hiding:** An object can hide the information that should remain unknown to other objects. Other information can be exchanged through the object's **methods**.
- **Advantages**
  - Messages are highly flexible since:
    - The sender and receiver can be of the same type, or not.
    - The sender and receiver can be on the same machine, or not (through network).

Encapsulation

封裝

Encapsulation =

Hide private data +

Provide public access interface.

- **Encapsulation** limits class data accesses and simplify tracking, debugging, and maintenance.

When Implemented in OOP…

Instantiate →

- “Type ⇒ **class**” ⇔ “Object ⇒ **class instance**”
  - State ⇒ **attributes**, **fields**, or **member variables** 成員變數
    - 靜態 成員變數 **Static member variables** are shared by all instances of the same class, e.g., we all live on earth.

      就是類別的成員變數：同個類別的成員共享的變數
    - *A class can have the instance of other classes as its member variables.*

      不是這個類別的都無法存取
  - 成員函數，也就是 method.

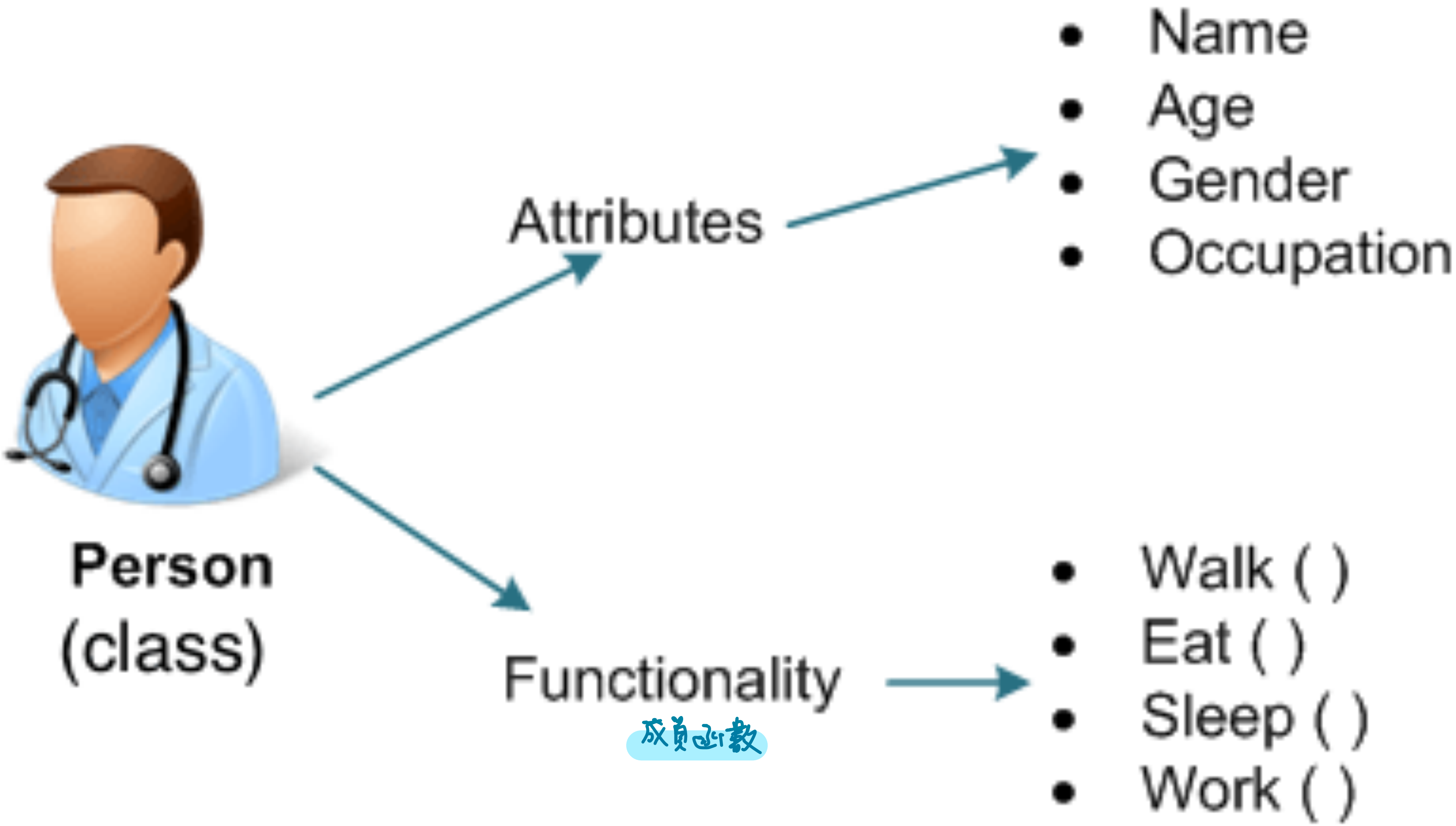
    Behavior ⇒ **methods** or **member functions**

    沒有物件的時候，只可以呼叫 1. 靜態方法 2. 類別方法
    - **Static methods / class methods** *does not rely on any class instance*, and *can access only the static member variables*.

Class vs. Procedure

- Split large programs into smaller **modules** to ease understanding, tracking, debugging, and maintenance.
- A **procedure** is a lower-level module that realizes a logical function.
- A **class** is a higher-level module that simulates all real-world objects *of the same type*.

Example of a Class



Person
(class)

Attributes

- Name
- Age
- Gender
- Occupation

Functionality

成員函數

- Walk ( )
- Eat ( )
- Sleep ( )
- Work ( )

https://dev.to/gbengelebs/introduction-to-classes-and-objects-in-object-oriented-programming-in-c-48pp

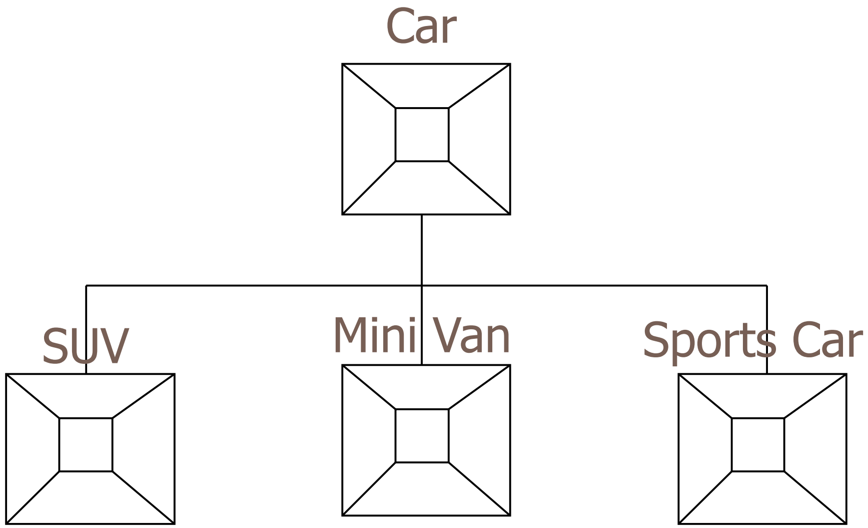
繼承關係

Inheritance: Is-a Relationship

- A university student is a student.
- All students learn. Thus any university students learn.
- However, elementary-school students *do not* select courses.
- From this perspective, roughly speaking, *a university student behaves like a normal student, plus some extra behaviors*.
- The **is-a relationship**, a.k.a. **inheritance,** is realized by a class UniversityStudent, which **inherits** the class Student to clone all behaviors of Student *by default*.
  - We may change the Student's default behavior if necessary.
- **[Exercise] Can you give 3 examples for is-a relationship?**

Subclass and Superclass

- If a class UniversityStudent inherits another class Student, we say that:
  - Student is the **superclass** or **supertype**.
  - UniversityStudent is the **subclass** or **subtype**.



- Besides inheritance, do we have any other relationships?

https://web.csulb.edu/~mopkins/cecs277/index.shtml ⁊ https://www.amazon.com/UBOXES-Moving-Inches-Bundle-BOXBUNDLAR12/dp/B007PBKMKK

Composition: Owns-a Relationship

- A company owns a CEO. Once the company is shut down, the CEO is fired and no longer exists.
- This is called an **owns-a relationship**, a.k.a. **composition**.
- **[Exercise] Can you give 3 examples for owns-a relationship?**

Aggregation: Has-a Relationship

- A student enrolls a B.S. program of a university.
- We can say that the university has the student.
- If the university is shut down, the student is still a student and is simply redirected to other universities to continue her or his journey.
- The student is not destroyed even if the university is shut down.
- This weaker relationship is called **aggregation**, a.k.a. **has-a relationship**.
- **[Exercise] Can you give 3 examples for has-a relationship?**

Override

- A subclass can simply inherit its superclass' default behaviors, or change it through **overriding**, which replaces the default behavior with a more appropriate one.
  - Example: A UniversityStudent can take course remotely, but this is not suitable for some Students.
- **2 similar concepts**
  - **Function overloading**: Create many functions with the same identifier but different parameter type lists, with related but different semantics.
  - **Class overriding**: A subclass overwrites the default behavior of the superclass with a different behavior.
- **[Exercise] Can you give 3 examples for function overloading and class overriding?**

Abstract Classes

- All instances of Students must be one among KinderGartenStudent, ElementarySchoolStudent, JouniorHighSchoolStudent, SeniorHighSchoolStudent, and UniversityStudent.
- In other words, there should be no actual instances of Student that do not belong to any subclass of Student.
- The Student class should not be directly instantiated; such class is called an **abstract class**.
- **[Exercise] Can you give 3 examples of abstract classes?**

Polymorphism

- A UniversityStudent combines all common behaviors of Student and its unique behaviors.
- Furthermore, the unique behaviors of UniversityStudent can *internally* interact with the behaviors of Students.
- This is called **polymorphism**.
- Story could become complex in **multiple inheritance**, with which *a subclass inherits more than one superclasses*.
  - **[Exercise] Can you give 3 examples of multiple inheritance?**

Procedural vs. Object-oriented Programming

- **Procedural programming** defines the flow of necessary computation steps to solve a problem.
  - **Bottom-up approach**: Write small components first and combine them into a large system.
  - **Top-down approach**: Break the large design problem into multiple parts, solving one at a time.
- **Object-oriented programming** attempts to define the roles (“*objects*”) involved in an ecosystem, along with their internal behaviors and external interactions.

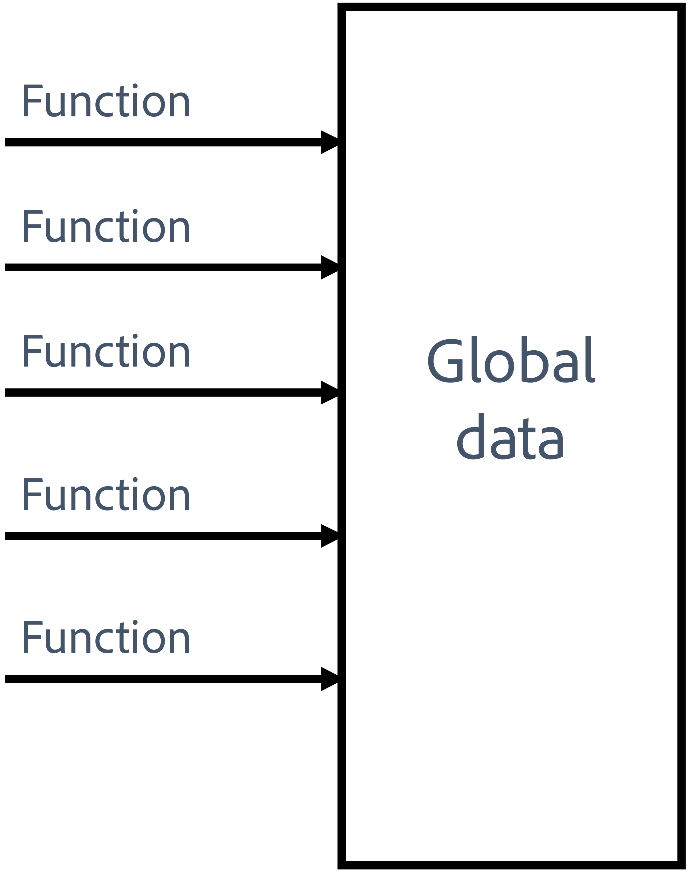
https://web.csulb.edu/~mopkins/cecs277/index.shtml尢https://www.amazon.com/UBOXES-Moving-Inches-Bundle-BOXBUNDLAR12/dp/B007PBKMKK

Advantages of OOP (Against Procedural Programming)

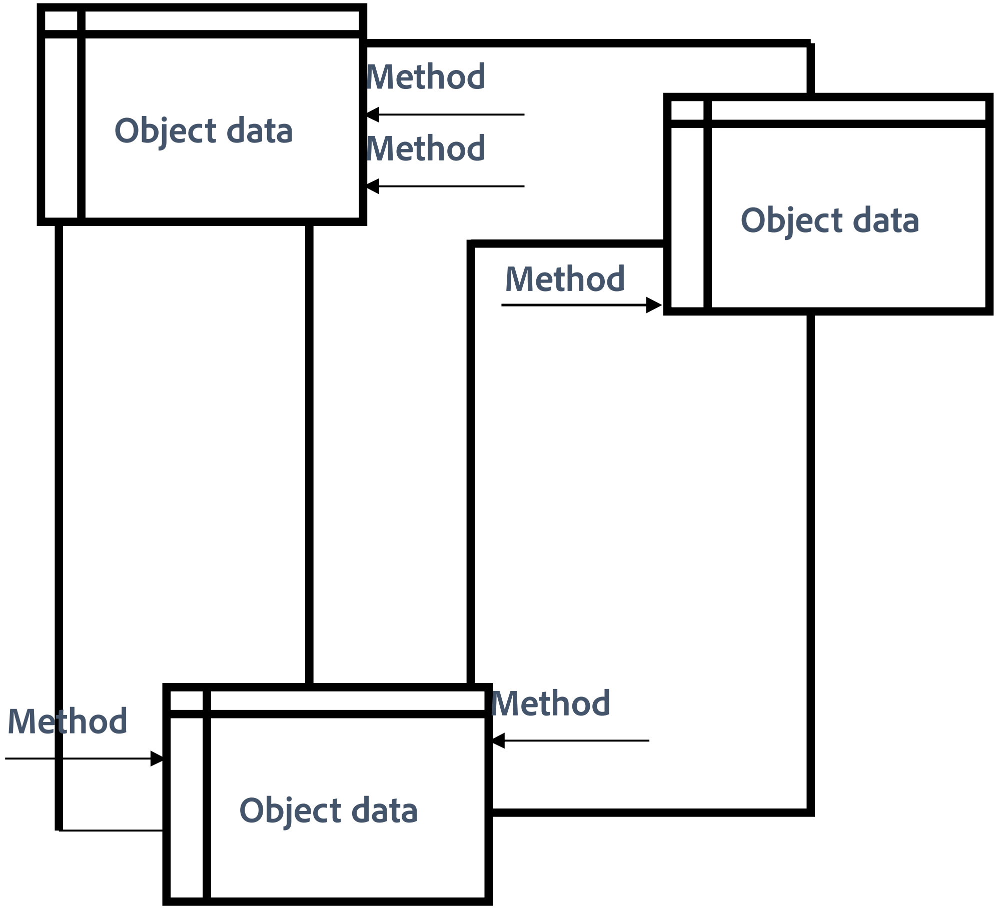
- Class is a powerful construct for connecting different logical parts of a system and making them to collaborate.
- Class can be developed and tested even before the whole system is ready. This significantly boosts the development process.
- Encapsulation and access control allow more programmers to work together without accidentally affecting each other.

Case Study: Finding a Data-Related Bug

- Procedural programming



- Object-oriented programming



https://web.csulb.edu/~mopkins/cecs277/index.shtmlㄤhttps://www.amazon.com/UBOXES-Moving-Inches-Bundle-BOXBUNDLAR12/dp/B007PBKMKK

Singleton

- Concerning the ecosystem of schools in Taiwan, there is a single instance of MOE (Ministry of Education) which interacts with all Universities.
- **Singleton classes** can be instantiated for a limited number of times, which is often but not limited to 1.
- *How do we guarantee that the maximum number of instances of a singleton class will not be exceeded?*
- **[Exercise] Can you give 3 examples of singleton classes?**

https://skyyen999.gitbooks.io/-study-design-pattern-in-java/content/singleton.html

Basic Principle: SOLID

- Single responsibility
- Open/close principle (OCP)
- Liskov substitution principle (LSP)
- Interface segregation (隔離) principle (ISP)
- Dependency Inversion Principle (DIP)

https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design#interface-segregation-principle

Single Responsibility

- *A class should have one and only one reason to change*, meaning that a class should have only one job.
- If there are multiple jobs of a class, each job should be assigned to a dedicated class, which will then be attributed to an owner class through composition or aggregation.

Open/close Principle (OCP)

- *Objects or entities should be open for extension but closed for modification.*
- This means that a class should be extendable without modifying the class itself.
- How to do this?
    - **Add a container class** that has the to-be-extended class as its member: composition or aggregation.
    - **Inherit the to-be-extended class**: inheritance/specialization.

Liskov Substitution Principle (LSP)

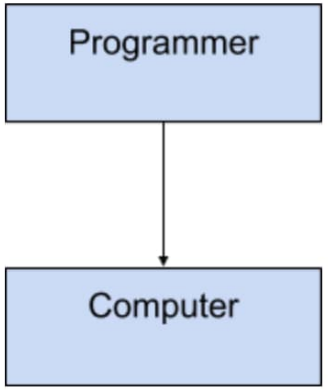
- Let $q(x)$ be a property provable about objects of x of type T. Then $q(y)$ should be provable for objects y of type S where S is a subtype of T.
- *Every subclass or derived class should be substitutable for their base or parent class.*
- Counter examples (Why?)
  - ToyCar seems not an appropriate subclass of Car.
  - Square seems not an appropriate subclass of Rectangle.

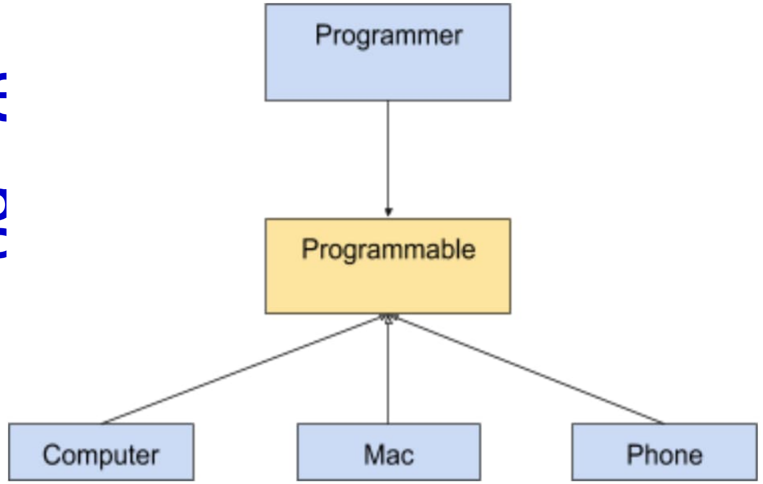
Interface Segregation Principle (ISP)

- *Use interfaces to separate different functionalities.*
- Technically, an interface is a class without data members. However, conceptually, the interface has very different usages from the class does.
- A client should never be forced to implement an interface that it doesn’t use, or clients shouldn’t be forced to depend on methods they do not use.

Dependency Inversion Principle (DIP)

- **Low flexibility**: High-level objects directly contact low-level ones.
- **High flexibility with DIP**: Both high- and low-level objects depend on an interface, so that new high- or low-leve objects can be added without affecting existing objects.





https://www.jyt0532.com/2020/03/24/dip/

Thank You Very Much!

Q&A?