fit@hcmus

Object-Oriented Programming

Introduction to OOP

Tran Duy Hoang

Programming Paradigms

- Fundamental styles or approaches to programming
- Provide different ways to design and structure code

- Types of programing paradigms:
  - Procedural Programming
  - Object-Oriented Programming
  - Parallel & Concurrent Programming
  - Functional Programming
  - Logic Programming
  - Constraint-Based Programming
  - Reactive Programming

Procedural vs. Object-Oriented Programming

- Problem: nấu món thịt kho trứng + rau muống xào.

Action
Lặt
Luộc
Ướp
Kho
Xào
Bóc vỏ

Procedural
Ướp (Thịt)
Luộc (Trứng)
Lặt (Rau)
Bóc vỏ (Trứng)
Kho (Thịt, Trứng)
Xào (Rau)

Object Oriented
Trứng. Luộc( )
Trứng. Bóc vỏ( )
Rau. Lặt( )
Rau. Xào( )
Thịt. Ướp( )
Thịt. Kho(Trứng)

Materials
Thịt
Trứng
Rau

Procedural:
- Action first.
- Function + Data.
(Verb) + (Object)

Object Oriented:
- Data first.
- Data triggers function.
(Object) does (Verb)
→ Change your thinking!!

Procedural vs. Object-Oriented Programming

- Procedural Programing
  - Functions & procedures
  - Data is separate from functions
  - Less secure (global variables)
  - Code reuse via functions

- Object-Oriented Programming
  - Objects & classes
  - Data and methods are bundled
  - More secure (encapsulation)
  - Code reuse via classes (inheritance)

Object-Oriented Programming

- Organize code around **objects** rather than **procedures**
- Objects encapsulate both
  - data (attributes)
  - behavior (methods)
- Key principles
  - Encapsulation – Data hiding
  - **Abstraction** – Hiding complexity
  - Inheritance – Code reusability
  - Polymorphism – Multiple forms

Object & Class

- Basic units of programs: variables, functions
- Procedural programming = function + variables

  → not easy to create abstract program

- Object-oriented programming = variable triggers function

  → need a new kind of unit

Special unit: Object!!!

Object & Class

- Object is a special variable, containing
  - Attribute: data of object
  - Method: functions of object

Function + Structure = Object

Object & Class

- Class is a blueprint for creating objects, descripting
  - Attributes (data members)
  - Methods (functions)

- An object is a specific **instance** of a class

Variable ~ Type

Struct variable ~ Struct type

Object ~ Class

Exercise 2.1

- You are designing a simple **Library Management System** where user can borrow books.

1. Identify the main classes required for the system
2. Determine the attributes and methods of each class

Exercise 2.2

- You are designing a simple **Hotel Booking System** where guests can book rooms, make payments, and check their reservations.

  1. Identify the main classes required for the system
  2. Determine the attributes and methods of each class

Exercise 2.3

- You are designing a simple **Online Shopping System** where users can browse products, place orders, and make payments.

1. Identify the main classes required for the system
2. Determine the attributes and methods of each class

Object Usage

- How to use object in C++
  - Declare class (file .h)
  - Implement class (file .cpp)
  - Create object from class (main() function)

Example: Object Usage

```
// Declare class (Fraction.h)

class Fraction
{
private:
    int num;
    int denom;

public:
    Fraction add(Fraction p);
};
```

```
// Declare struct (Fraction.h)

struct Fraction
{
    int num;
    int denom;
};

Fraction add(Fraction p1, Fraction p2);
```

Example: Object Usage

```
// Implement class (Fraction.cpp)

Fraction Fraction::add(Fraction p)
{
    Fraction sum;

    sum.num = num * p.denom + denom * p.num;
    sum.denom = denom * p.denom;

    return sum;
}
```

```
// Implement method (Fraction.cpp)

Fraction add(Fraction p1, Fraction p2)
{
    Fraction sum;

    sum.num = p1.num * p2.denom + p1.denom * p2.num;
    sum.denom = p1.denom * p2.denom;

    return sum;
}
```

Example: Object Usage

```
// Use object (main.cpp)

#include "Fraction.h"

int main()
{
    Fraction p1;
    Fraction p2;

    Fraction p3 = p1.add(p2);
}
```

```
// Use struct varible (main.cpp)

#include "Fraction.h"

int main()
{
    Fraction p1;
    Fraction p2;

    Fraction p3 = add(p1, p2);
}
```

Types of Scope

- Local Scope (Block Scope)
  - A variable declared inside a function or block
  - It cannot be accessed outside that block
- Global Scope
  - A variable declared outside all functions
  - Accessible from any function in the same file
- Function Scope
  - Functions are globally accessible
- Class Scope (Member Scope)
  - Attributes and methods of an object have class scope

Class Scope

Keywork	Scope
private	Inside class only
public	Inside and outside class
protected	Inside class and children class

```
class A
{
private:
    int x;
public:
    int y;
public:
    int getX();
private:
    void calculate();
};
```

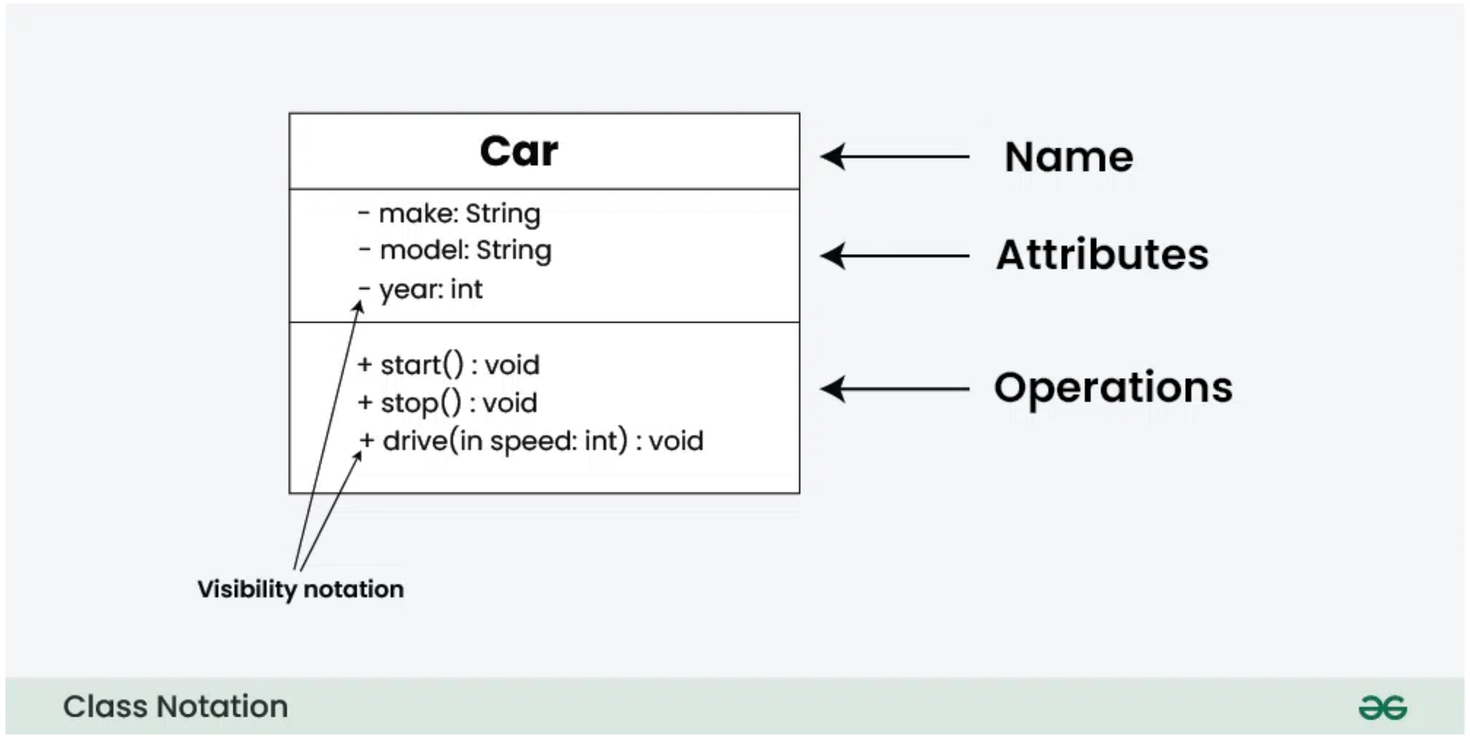
```
int main()
{
    A obj;
    int x = obj.x;  // Wrong
    obj.x = 1;      // Wrong

    int y = obj.y;  // Right
    obj.y = 1;      // Right

    int t = obj.getX(); // Right
    obj.calculate();    // Wrong
}
```

Black Box Principle

- An object acts as a “black box”
  - hide internal data and logic (private variables & methods)
  - expose only the necessary functions (public methods)

```
class Fraction
{
private:
    int num;
    int denom;
public:
    Fraction add(Fraction p);
    Fraction reduce();
};
```

Class Diagram



Class Notation

Exercise 2.4

- Construct class Fraction in C++
  - Attribute: numerator, denominator
  - Methods:
    - input: enter fraction from keyboard
    - output: print fraction to screen
    - getNum/setNum: get/update numerator of fraction
    - getDenom/setDenom: get/update denominator of fraction
    - reduce: return the reduction of fraction
    - inverse: return the inversion of fraction
    - add: return the sum of two fractions
    - compare: return the comparison result of two factions
      - 0: first = second, -1: first < second, 1: first > second

Exercise 2.5

- Construct class Student in C++
  - Attribute: name, math score, physics score, chemistry score
  - Methods:
    - input: enter student information from keyboard
    - output: print student information to screen
    - getName/setName: get/update name of student
    - getMath/setMath: get/update math score of student
    - getPhysics/setPhysics: get/update physics score of student
    - getChem/setChem: get/update chemistry score of student
    - calculateGPA: return GPA of student
      - GPA = (math + physics + chem) / 3
    - grade: return student grade
      - A: GPA >= 9.0, B: GPA >= 7.0, C: GPA >=5.0, D: GPA < 5

Exercise 2.6

- Construct class Array in C++
  - Attribute: array of integers, size of array
  - Methods:
    - input: enter array size and array elements
    - output: print array elements to screen
    - getSize/setSize: get/update size of array
    - getElement/setElement: get/update element at specific index
    - find: look for a value and return found index
      - -1 if not found
    - sort: sort array, the sort criteria can be customized