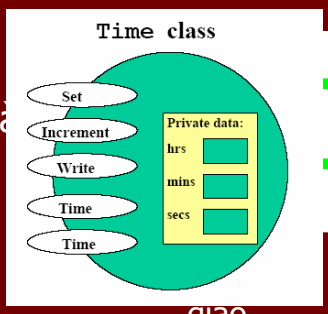
Chương 3: Thực hiện ẩn Khởi tạo và hủy bỏ đối tượng

Huỳnh Quyết Thắng
Cao Tuấn Dũng
Bộ môn CNPM

Vai trò của thực hiện ẩn

- Khi viết chương trình thông thường có hai trường hợp xảy ra: (1) chúng ta viết thư viện và (2) chúng ra sử dụng thư viện
- Trong trường hợp (1): chúng ta không muốn cho các LTV sử dụng thư viện được truy nhập/can thiệp vào các phần lõi của thư viện
- Trong trường hợp (2): chúng ta không cần quan tâm tới phần lõi của thư viện, chúng ta chỉ cần quan tâm giao diện của phần được sử dụng không bị thay đổi
- Giải pháp: Các cơ chế thực hiện/khai báo ẩn (hidden implementation)

Đối tượng và giao tiếp với thế giới bên ngoài



Bên ngoà

giao diện

Phần Lõi/nhân

Từ khóa xác định thuộc tính truy nhập (access specifier)

- C++/Java quy định 3 từ khóa xác định tính chất của các thuộc tính của đối tượng: public, private, protected
- Public: quy định các thuộc tính của đối tượng là được phép sử dụng/truy nhập từ bất kỳ ai/đối tượng nào
- Private: quy định các thuộc tính của đối tượng không được phép sử dụng từ các đối tượng bên ngoài. Đây như là bức tường đối với các LTV bên ngoài đối tượng

Từ khóa xác định thuộc tính truy nhập (access specifier)

- Protected: từ khóa đặc biệt quy định tính chất của các đối tượng chỉ được phép truy nhập của các đối tượng bên ngoài lớp nhưng phải thuộc về một lớp nào đó kế thừa của lớp hiện thời
- Khi khai báo lớp 3 từ khóa này có ý nghĩa đánh dấu tính chất của các thuộc tính (hàm thành phần và dữ liệu thành phần) kể từ lúc gặp các từ khoá tương ứng cho tới khi gặp từ khóa khác
- Mặc định: private
- Thứ tự xuất hiện các từ khóa là tùy ý.

Ví dụ minh họa

```
class Square
  {
     private:
        int side;
     public:
        void setSide(int
  s)
        { side = s; }
        int getSide()
        { return side; }
  };
```

```
int main()
{
  Square s1;
  s1.setSide(10);
  s1.side = 15;
}
```

Ví dụ minh họa

```
public class Car
{
...
private String model;
public int mileage;
}
...
Car aCar = new Car();
aCar.mileage = 23000; // ok
System.out.println(aCar.mileage); // ok
aCar.model = "VW Beetle"; // error
System.out.println(aCar.model); // error
```


Java

```
class Sundae {
    private Sundae() {}
    static Sundae makeASundae() {
      return new Sundae();
    }
  }

  public class IceCream {
    public static void main(String[] args) {
      //! Sundae x = new Sundae();
      Sundae x = Sundae.makeASundae();
    }
  }
```

Contructor là phương thức private

Không dùng được toán tử new. Việc tạo ra một đối tượng phải được thông qua phương thức:

makeASundae()

Các phương thức mà kết quả của nó chỉ để phục vụ cho các phương thức cùng lớp (helper method) thì nên để chế độ private

Cấu trúc lớp thường dùng

```
class X {
  void private_function();
  int internal_representation;
public:
  void interface_function();
};
```

Các phương thức mà kết quả của nó chỉ để phục vụ cho các phương thức cùng lớp (helper method) thì nên để chế độ private

Java: Truy nhập lớp, package

- Quy định quyền truy nhập đến một lớp của một thư viện
- Từ khóa public class X: lớp X được nhìn thấy từ mọi nơi bên ngoài

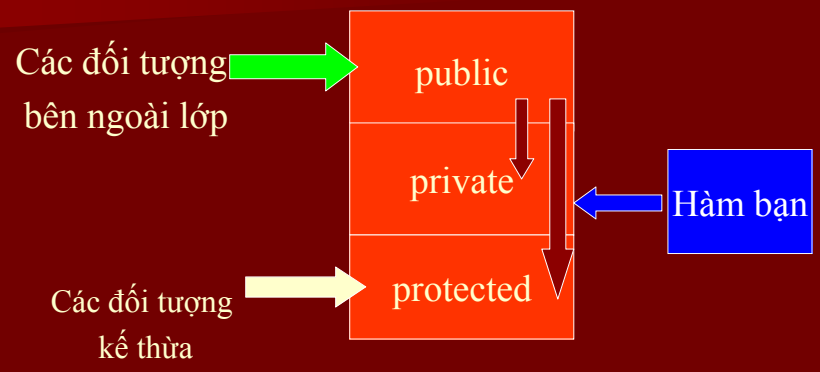
```
public class Widget { …
import mylib.Widget;
  hoặc
import mylib.*;
```

- Ngược lại, lớp X chỉ được nhìn thấy bởi các lớp trong cùng package với nó.

C++: Cơ chế hàm bạn

- Muốn truy nhập tới các dữ liệu thành phần private của một đối tượng theo quy định chỉ có các hàm thành phần của lớp đó
- Chúng ta muốn có một hàm đặc biệt là hàm "bạn" của lớp, nó không phải là hàm thành phần của lớp nhưng được quyền truy nhập các thành phần dữ liệu riêng (private) của đối tượng
- Tính chất đặc biệt của hàm bạn:
  - được quyền truy nhập các thành phần dữ liệu riêng (private)
  - không phải là hàm thành phần

Hàm bạn: bản chất, ý nghĩa, tồn tại



Hàm bạn: bản chất, ý nghĩa, tồn tại

- Các đặc điểm của hàm bạn:
  - Phải được định nghĩa với từ khóa friend trong lớp mà hàm đó muốn trở thành hàm bạn
  - Trong đối số của hàm bạn phải có ít nhất một đối tượng thuộc lớp (để có thể truyền biến truy nhập các thành phần của đối tượng)
  - Vị tri khai báo hàm bạn trong lớp là tuỳ ý và không bị ảnh hưởng/chi phối bởi các từ khóa private/public/protected

Hàm bạn: bản chất, ý nghĩa, tồn tại

- Hàm bạn không phải là hàm thành phần nhưng có quyền truy nhập các dữ liệu thành phần
- Tư tưởng về hàm bạn đi ngược lại với tư tưởng đóng gói (encapsulation) thể hiện trong thực hiện ẩn
- Bản chất C++ là ngôn ngữ lập trình hướng đối tượng lai tạp (hybriđ), trong đó hàm bạn được thêm vào để tăng tính mềm dẻo của kỹ thuật lập trình trong ngôn ngữ này

Các dạng hàm bạn

- Có nhiều kiểu hàm bạn:
  - Hàm tự do là bạn của một lớp
  - Hàm thành phần của một lớp là bạn của lớp khác
  - Hàm bạn là bạn của nhiều lớp
  - Tất cả các hàm thành phần của một lớp là hàm bạn của lớp khác

Hàm tự do là bạn của một lớp

```
class A
      {
  private:
   // Khai báo các thuộc tính
  public:
        ...
 // Khai báo các hàm bạn của
// lớp A
      friend void f1(...);
     friend double f2(...);
     friend A f3(...);
     ...
     };
```

```
// Xây dựng các hàm f1, f2, f3
void f1(...)
    {
        ...
    }
double f2(...)
    {
        ...
    }
A f3(...)
    {
        ...
    }
```

So sánh phương thức và hàm bạn: Cộng hai số phức: phương thức

```
class SP
     {
  private:
       double a; // Phần thực
       double b; // Phần ảo
  public:
      SP cong(SP u2)
      {
           SP u:
           u.a = this→a + u2.a;
           u.b = this→b + u2.b;
           return u;
      }
     };
```

Cách dùng :

```
SP u, u1, u2;
u = u1.cong(u2);
```


Cộng hai số phức: hàm bạn

```
class SP
     {
  private:
       double a; // Phần thực
       double b; // Phần ảo
  public:
       friend SP cong(SP u1, SP
  u2)
      {
           SP u:
           u.a = u1.a + u2.a;
           u.b = u1.b + u2.b;
           return u;
      }
     };
```

Cách dùng :

SP u, u1, u2;
u = cong(u1, u2);

Hàm thành phần là bạn của lớp khác

```
class A;
class B {
    ..............
  int f(A);
    ..........
};
class A {
  ...........
friend int B::f (A);
  ......
};
```

```
int B::f(A) {
  ....
}
```

- *int f(A) là hàm thành phần của lớp B*
- *int f(A) là hàm bạn của lớp A*

Hàm là bạn của nhiều lớp

- Khi một hàm là bạn của nhiều lớp, thì nó có quyền truy nhập tới tất cả các thuộc tính của các đối tượng trong các lớp này.

```
class A; // Khai báo trước lớp A
class B; // Khai báo trước lớp B
// Định nghĩa lớp A
class A {
  // Khai báo hàm f là bạn của lớp A
  friend void f(...);
};
```

Hàm là bạn của nhiều lớp

```
class B{
   // Khai báo hàm f là bạn của lớp B
   friend void f(...);
};

// Xây dựng hàm f
void f(...){
   ...
}
Hàm f là bạn của cả lớp A và B
```

- Chương trình sau đây minh họa cách dùng hàm bạn (bạn của một lớp và bạn của nhiều lớp). Chương trình đưa vào 2 lớp VT (véc tơ), MT (ma trận) và 3 hàm bạn để thực hiện các thao tác trên 2 lớp này

Nhân ma trận và vec tơ

```
#include<iostream>
#include<cmath>
class VT;
class MT;
class VT;
{
private:
     int n;
   double x[20]; // Toa do cua diem
public:
   void nhapsl();
   friend void in(const VT&x);
   friend VT tich(const MT&a, const VT&x);
   };
```

Nhân ma trận và vec tơ

```
class MT
   {
private:
      int n;
      double a[20][20];
public:
      friend VT tich(const MT&a, const VT&x);
      friend void in(const MT&a);
   void nhapsl();
   };

void VT::nhapsl()
   {
      cout<<"\n Cap vecto = ";
      cin>>n;
      for (int i=0; i<n; ++i)
   {
      cout<<"\n Phan tu thu "<<i<<" = ";
      cin>>x[i];
   }
   }
```


Nhân ma trận và vec tơ

```
void MT::nhapsl()
   {
cout<<"\n Cap ma tran =
   ";
cin>>n;
for (int i=0; i<n; ++i)
 for (int j=0; j<n; ++j)
  {
      cout<<"\n Phan tu
  thu hang "<<i<<" cot
  "<<j<<" = ";
      cin>>a[i][j];
    }
   }
```

```
VT tich(const MT&a, const VT&x)
  {
   VT y;
   int n=a.n;
   if (n!=x.n)
   return x;
   y.n=n;
   for (int i=0; i<n; ++i)
          {
           y.x[i]=0;
           for (int j=0; j<n; ++j)
             y.x[i]+=a.a[i][j]*x.x[j];
          }
   return y;
  }
```

Nhân ma trận và vec tơ

```
void in(const VT &x)
   {
      cout<<"\n";
      for (int i=0; i<x.n; ++i)
      cout<<x.x[i]<<" ";
   }
void in(const MT&a)
   {
      for (int i=0; i<a.n; ++i)
   {
         cout<<"\n";
         for (int j=0; j<a.n; ++j)
        cout<<a.a[i][j]<<" ";
   }
   }
```


Nhân ma trận và vec tơ

```
void main()
   {
      MT a; VT x,y;
      a.nhapsl();
      x.nhapsl();
      y=tich(a,x);
      cout<<"\nMa tran A : ";
      in(a);
      cout<<"\n\nVecto x : ";
      in(x);
      cout<<"\n\nVec y = Ax : ";
      in(y);
}
```

Lớp bạn

- Tất cả các hàm của lớp (B) là bạn của lớp khác (A)
  - Đây là trường hợp tổng quát trong đó có thể khai báo lớp bạn bè với các hàm. Mọi vấn đề sẽ đơn giản hơn nếu ta đưa ra một khai báo tổng thể để nói rằng tất cả các hàm thành phần của lớp B là bạn của lớp A. Muốn vậy ta đặt trong khai báo lớp A chỉ thị: ***friend class B***;

Lớp bạn

```
class A     {
   ...
   friend class B; // Lớp B là bạn của lớp A
   friend class C; // Lớp C là bạn của lớp A
   ...
};
class B     {
   ...
   friend class A; // Lớp A là bạn của lớp B
   friend class C; // Lớp C là bạn của lớp B
   ...
};
class C     {
   ...
   friend class A; // Lớp A là bạn của Lớp C
   friend class B; // Lớp B là bạn của Lớp C
   ...
};
```

Java: Khái niệm friend???

- Java là ngôn ngữ lập trình HĐT "thuần", khác với C++ là một ngôn ngữ lai
- Trong Java không có khái niệm hàm bạn

Lớp: Các phương thức cơ bản

- Thông thường, một lớp bao giờ cũng có một số dạng phương thức:
  - Thiết lập, khởi tạo (constructor) và hủy bỏ (destructor) đối tượng
  - Thay đổi trạng thái đối tượng (mutator –setter): Các phương thức cho phép thiết lập, thay đổi giá trị của thành phần dữ liệu
  - Truy nhập giá trị trạng thái (accessor – getter): Các phương thức chỉ truy nhập giá trị thành phần dữ liệu mà không thay đổi chúng. Các phương thức này còn có tên gọi khác là các phương thức truy vấn (queries)

```
class Automobile {
  public:
    Automobile();
    void Input();
    void set_NumDoors( int doors );

    void Display();
    int get_NumDoors();

  private:
    string Make;
    int    NumDoors;
    int    NumCylinders;
    int    EngineSize;
};
```


Phương thức Get Set

restricted access: *private* members are *not externally accessible*; but we need to know and modify their values

set methods: *public* methods that allow clients to *modify private* data; also known as *mutators*

```
public class Time {
    private int hour;
    private int minute;
    private int second;

    public Time () {
        setTime(0, 0, 0);
    }

    public void setHour (int h) { hour = ( ( h >= 0 && h < 24 ) ? h : 0 ); }

    public void setMinute (int m) { minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); }

    public void setSecond (int s) { second = ( ( s >= 0 && s < 60 ) ? s : 0 ); }

    public void setTime (int h, int m, int s) {
        setHour(h);
        setMinute(m);
        setSecond(s);
    }

    public int getHour () { return hour; }

    public int getMinute () { return minute; }

    public int getSecond () { return second; }
}
```

get methods: *public* methods that allow clients to *read private* data; also known as *accessors*

Phương thức Set (Mutator)

Thường chứa toán tử gán, và nhận tham số từ bên ngoài

```
class Person{
   private:
     int age;

   public:
    void setAge(int newAge){
        if ((newAge > 0) && (newAge < 150))
            age = newAge;
    }
}
```


Phương thức Set

- Nếu các phương thức get/set chỉ có nhiệm vụ cho ta đọc và ghi giá trị cho các thành viên dữ liệu, quy định các thành viên private để được ích lợi gì?
- Ngoài việc bảo vệ các nguyên tắc đóng gói, ta còn cần kiểm tra xem giá trị mới cho thành viên dữ liệu có hợp lệ hay không
  - Ví dụ, cần đảm bảo rằng điểm trung bình của sinh viên không bị gán về số âm.
- Sử dụng phương thức truy vấn cho phép ta thực hiện việc kiểm tra trước khi thực sự thay đổi giá trị của thành viên.

Ví dụ: Phương thức Set

- Lớp sinh viên có phương thức Điểm trung bình

```
int Student::setGPA(double newGPA)
{
    if ((newGPA >= 0.0) && (newGPA <= 4.0))
    {
        this->gpa = newGPA;
        return 0; // Return 0 to indicate success
    }
    else return -1; // Return -1 to indicate failure

}
```


Phương thức Get (Truy vấn)

- Các phương thức truy vấn (query method, accessor) là các phương thức dùng để hỏi về giá trị của các thành viên dữ liệu của một đối tượng
- Có nhiều loại câu hỏi truy vấn có thể:
  - truy vấn đơn giản (*"giá trị của x là bao nhiêu?"*)
  - truy vấn điều kiện (*"thành viên x có lớn hơn 10 không?"*)
  - truy vấn dẫn xuất (*"tổng giá trị của các thành viên x và y là bao nhiêu?"*)
- Đặc điểm quan trọng của phương thức truy vấn là nó không nên thay đổi trạng thái hiện tại của đối tượng
  - không thay đổi giá trị của thành viên dữ liệu nào.

Phương thức Get

- Đối với các truy vấn đơn giản, quy ước đặt tên phương thức: tiền tố "get", tiếp theo là tên của thành viên

```
// query returns value of member x
int getX();
// query returns value of member size
int getSize();
```

- Các loại truy vấn khác nên có tên có tính mô tả
- Truy vấn điều kiện nên có tiền tố "is"

```
int Foo::getXPlusY() { return x + y; }
bool Foo::isXPositive() { return x > 0; }
```


Khởi tạo và hủy bỏ đối tượng

1. Ý nghĩa của quá trình khởi tạo và huỷ bỏ đối tượng
2. Quá trình khởi tạo: các hàm thiết lập (constructor)
3. Quá trình huỷ đối tượng: hàm huỷ (destructor)
4. Các đặc điểm của hàm khởi tạo và hàm huỷ
5. Hàm khởi tạo mặc định
6. Các trường hợp đối tượng được tao ra và giải phóng

Ý nghĩa

- Mỗi đối tượng khi tồn tại và hoạt động được hệ điều hành cấp pháp một vùng nhớ (theo giao diện của lớp) để lưu lại các giá trị của dữ liệu thành phần
- Khi tạo ra đối tượng hệ điều hành sẽ gán luôn cho các dữ liệu thành phần này các giá trị khởi tạo tuỳ theo mong muốn của LTV quy định bằng các lệnh khai báo biến-đối tượng
- Ngược lại khi kết thúc vòng đời của đối tượng cần phải giải phóng hợp lý tất cả các bộ nhớ đã cấp phát cho đối tượng (do LTV hoặc Trình BD)

Quá trình khởi tạo: Các hàm thiết lập Constructor

- Các quá trình gán dữ liệu hay huỷ dữ liệu này phải được thực hiện tự động trước khi người lập trình có thể tác động lên đối tượng
- Hàm thiết lập là một hàm đặc biệt. Hàm này được gọi tự động mỗi khi có một đối tượng mới được tạo ra. Chức năng của hàm thiết lập là khởi tạo các giá trị của các thành phần dữ liệu của đối tượng hay xin cấp phát bộ nhớ cho các thành phần bộ nhớ động.

Ví dụ constructor

```
class Square
   {
      public:
         Square();    // prototype
   ...
   };
   Square::Square() // heading
   {
         side = 1;
   }
```


Constructor

- Constructor có vai trò đảm bảo thành phần dữ liệu của một đối tượng được khởi tạo, không ở trạng thái chưa xác định.
- Các trường hợp Constructor được gọi:
  - Khi khai báo đối tượng
  - Truyền đối tượng dưới dạng tham trị
  - Cấp phát động
- Không có tham số: Constructor mặc định
- Có tham số: Có nhiều constructor trùng tên. Cần phân biệt qua danh sách tham số

c++ constructor

```
class Foo {
    public:
    Foo(); // Default constructor
    Foo(int x); // Overloaded constructor
    Foo(string s); // Overloaded constructor
    …
};
```


Hàm hủy: Destructor (C++)

- Ngược lại với quá trình khởi tạo đối tượng, khi giải phóng đối tượng chúng ta phải giải phóng toàn bộ bộ nhớ đã được cấp phát cho đối tượng. Chức năng của hàm huỷ sẽ thực hiện vai trò này:
- Ví dụ:

```
class  A {
 int n;
 public:
   A();  //constructor
  ~A();  // destructor
 };
```

- Java: không dùng hàm hủy.

Hàm hủy

- Trước khi HDH giải phóng bộ nhớ đã cấp phát để lưu trữ các dữ liệu thành phần của đối tượng, sẽ thực hiện hàm huỷ. Vì vậy chúng ta lưu ý khi xây dựng các lớp, trong hàm huỷ chỉ cần giải phóng những gì mà HDH không tự động giải phóng cho chúng ta.
- Nguyên tắc cần lưu ý ở đây là:
  - Cần đặc biệt lưu ý tới các lớp trong đó có dữ liệu thành phần là các con trỏ
  - Không bỏ sót (không hiệu quả sử dụng bộ nhớ) và không giải phóng các bộ nhớ cấp phát hai lần (sẽ báo lỗi)

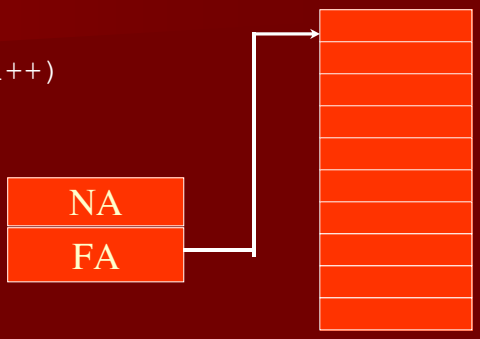
Quá trình hủy DT: hàm hủy

```
#include <iostream>
#include <conio.h>
#include <stdio.h>

class A
 { int NA;
   float *FA;
   public:
   A(int m);
   void display();
 };
```

```
A::A(int m)
{
 NA=m;
 FA = new float [m];
 for (int i=0; i<m;
 i++)
  { FA[i]=i*10.0; }
 }
```

Quá trình hủy DT: hàm hủy

```
void A::display()  {
for (int i=0; i<NA; i++)
     { cout << FA[i];
       cout << "  ";
     }
}

void main()
{ A A1(5);
  A1.display();
}
```



Giải phóng A1, mảng vẫn còn?
Giải quyết triệt để giải phóng bộ nhớ?

Hàm hủy

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>

class A
 { int NA;
   float *FA;
   public:
   A(int m);
   void display();
   ~A();
 };
```

```
A::A(int m)
 {
  NA=m;
  FA = new float
  [m];
  for (int i=0; i<m;
  i++)
   { FA[i]=i*10.0; }
 }
A::~A()
 { delete FA[];}
```

Đặc điểm hàm thiết lập

- Hàm thiết lập có cùng tên với lớp.
- Hàm thiết lập phải có thuộc tính public
- Hàm thiết lập không có thuộc tính trả về và không cần khai báo void
- Mỗi lớp có thể có nhiều hàm thiết lập trong cùng một lớp
- Được tự động gọi trên cơ sở tìm thấy hàm thiết đúng nhất với các đối số truyền vào khi đăng ký đối tượng

Đặc điểm hàm hủy

- Tên hàm huỷ bỏ bắt đầu bằng dấu ~ theo sau là tên lớp tương ứng.
- Hàm huỷ cần có thuộc tính public
- Mỗi lớp chỉ có duy nhất một hàm huỷ bỏ.
- Khi không định nghĩa hàm huỷ bỏ chương dịch tự động sinh một hàm huỷ ngầm định để lấp vào chỗ trống để đảm bảo nguyên tắc luôn luôn thực hiện hàm huỷ.
- Hàm huỷ bỏ không có giá trị trả về.

Constructor và Destructor mặc định

- Trên thực tế tất cả các chương trình dịch khi dịch các lớp nếu như trong các lớp này không định nghĩa một hàm thiết lập nào cả thì sẽ tự động thêm vào lớp đó một hàm khởi tạo mặc định ngầm định (theo quy định của chương trình dịch đó). Tương tự như vậy chương trình dịch sẽ làm cả với hàm huỷ.
- Nếu trong lớp có định nghĩa ít nhất một hàm thiết lập thì chương trình dịch sẽ tuân theo cách định nghĩa lớp mà không thêm gì cả.
- Tuy nhiên, nếu ta không định nghĩa constructor mặc định nhưng lại có các constructor khác, trình biên dịch sẽ báo lỗi không tìm thấy constructor mặc định nếu ta không cung cấp tham số khi tạo thể hiện.

Quay lại lớp Automobile

```
class Automobile {
  public:
    Automobile( );
    void Input( );
    void set_NumDoors( int doors );

    void Display( );
    int get_NumDoors( );

  private:
    string Make;
    int    NumDoors;
    int    NumCylinders;
    int    EngineSize;
};
```

Constructor mặc định

```
Automobile::Automobile( )
{
  NumDoors = 0;
  NumCylinders = 0;
  EngineSize = 0;
}
```


Constructor định nghĩa chồng

```
class Automobile {
  public:
  Automobile( );
  Automobile(int d, int c, int s);
  Automobile(string m, int d, int c,
    int s);
  ……
```

Constructor với đối số ngầm định

```
class Automobile {
public:
   Automobile();

   Automobile( string make, int doors,
       int cylinders = 4, int engineSize =
    2000 );

   Automobile( const Automobile & A );
   // copy constructor
```


Tự tham chiếu

- Chúng ta cần một phương tiện:
  - Cho phép truy nhập đến đối tượng hiện hành của lớp.
  - Cho phép một đối tượng "tham chiếu" đến chính nó. Quan trọng khi hàm thành phần thao tác trên hai hay nhiều đối tượng.
  - Xóa đi sự nhập nhằng giữa một biến cục bộ, tham số với thành phần dữ liệu của lớp
- Con trỏ this (trong C++), tham chiếu this (Java)

Tự tham chiếu: this

- Java:
  - this. hour = hour
- C++:
  - this->hour = hour
- Con trỏ this (C++) hay tham chiếu this có quan hệ mật thiết đến các phương thức Get Set và các phương thức khởi tạo.
  - C++: copy constructor, assignment operator
  - Java: lời gọi đệ quy this(…)
- Không dùng bên trong các phương thức tĩnh

C++

```
int point::coincide(point pt)   {
  return(this->x==pt.x && this-
  >y==pt.y);
}

void point::display()  {
   cout<<"Dia chi : "<<this<<"Toa do :
  "<<x<<" "<<y<<"\n";
}
```

Java

```
public class Person
{
...
public void setName(String name)
{
this.name = name;
}
...
private String name;
private int age;
}// end class Person
```


JAVA

recursive calls: `this(...)` calls the constructor with the designated parameters

terminology: constructors having *parameters instances* of the *class* they are instantiating are sometimes known as *copy constructors*

Overloaded constructors: all *constructors* have the *same name*, but they *must have different signatures* (*i.e.*, different number/type of parameters)

base case: as with all recursive calls, *there should always be a base case* (*i.e.*, a non-recursive constructor)

```
public class Time {
    private int hour;
    private int minute;
    private int second;

    public Time () {
        this(0, 0, 0);
    }

    public Time (int h) {
        this(h, 0, 0);
    }

    public Time (int h, int m) {
        this(h, m, 0);
    }

    public Time (int h, int m, int s) {
        setTime(h, m, s);
    }

    public Time (Time t) {
        this(t.getHour(), t.getMinute(), t.getSecond());
    }

    public void setTime (int h, int m, int s) {
        ...
    }

    ...
}
```

Câu hỏi, bài tập

- Các câu hỏi:
- 1. Ý nghĩa của thực hiện ẩn trong LTHDT
- 2. Phân tích vai trò và ý nghĩa của các từ khóa public, private, protected
- 3. Phân tích quá trình cấp phát bộ nhớ trong cho các biến-đối tượng thuộc các lớp
- 4. Nêu vai trò của các toán tử (ký pháp): ., ->, :: khi nào thì sử dụng chúng
- 5. Ý nghĩa của hàm bạn LTHDT. Nêu đặc điểm của hàm bạn.
- 6. Nêu các đặc điểm của các hàm khởi tạo.
- 7. Nêu các đặc điểm của các hàm huỷ.

Câu hỏi, bài tập

- Bài tập tuần 3:
- Chuyển đổi bài tập tuần 2 thành lớp trong đó dữ liệu thành phần là mảng và số phần tử trong mảng. Các hàm thành phần là các thao tác sắp xếp, tìm kiếm và thao tác vào dữ liệu được khai báo như hàm khởi tạo
- Xây dựng lớp Stack mô phỏng cấu trúc Stack với các hoạt động sau:
  - Khởi tạo stack
  - Thêm phần tử vào Stack
  - Lấy phần tử khỏi Stack
  - In danh sách các phần tử có trong Stack
  - Hàm bạn Inmax in ra phần tử có giá trị lớn nhất trong Stack