Phân tích thiết kế hướng đối tượng

Bài 2b: Một bài toán Quản lý

TS. Nguyễn Hiếu Cường
Bộ môn CNPM, Khoa CNTT
Trường ĐH GTVT
cuonggt@gmail.com

Hà Nội, 2017

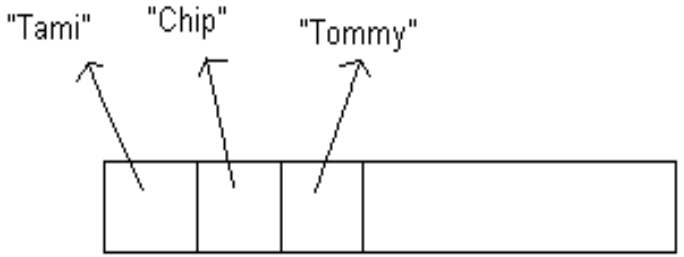
Mục đích bài này

- Ví dụ về một bài toán cao hơn mức độ "cơ bản" → nhu cầu phải có phân tích thiết kế

- Vận dụng các khái niệm hướng đối tượng thông qua việc cài đặt chương trình C++

Bài toán cần giải quyết

- Suzy có:
  - Một *chuồng* gồm nhiều *ngăn* để nuôi các con vật làm cảnh, mỗi ngăn chỉ nuôi 1 con, mỗi con vật đều có tên
  - Nhiều loại con vật khác nhau



- Suzy cần:
  - Xây dựng một chương trình để quản lý
- Chương trình có nhiệm vụ gì?
  - *Xác định danh tính* con vật trong từng ngăn: tên gì, loại gì?
  - *Thêm* con vật mới mới vào chuồng, *lấy* một con vật ra khỏi chuồng
  - Có thể sửa chữa, mở rộng chương trình một cách dễ dàng

Phân tích

- Có những đối tượng gì trong bài toán?
  - **Các con vật** (giả thiết Suzi chỉ nuôi mèo và chó)
  - **Chuồng** (gồm nhiều ngăn, mỗi ngăn nuôi một con)
- Các lớp cần xây dựng?
  - **Animal** (mô tả các đặc tính chung của các loại con vật)
  - **Cat** (con mèo) kế thừa lớp Animal
  - **Dog** (con chó) kế thừa lớp Animal
  - **Kennel** (chuồng nuôi các con vật)
- Trách nhiệm của từng lớp là gì?

Thiết kế

- Lớp Animal
  - Các đặc tính chung của một con vật, là cơ sở cho Cat và Dog
- Lớp Cat
  - Các đặc tính riêng của con mèo, ví dụ: tiếng "kêu" (kiểu của mèo)
- Lớp Dog
  - Các đặc tính riêng của con chó, ví dụ: tiếng "kêu" (kiểu của chó)
- Lớp Kennel
  - Các thao tác "thêm", "bớt" một con vật
  - "Liệt kê" danh sách các con vật đang có trong chuồng

Lớp Animal

```
class Animal {
protected:
  char* name;          // ví dụ chỉ có 1 đặc tính chung "name"
public:
  Animal() {  name= NULL;  }  // hàm tạo không đối
  Animal(char* name1)         // hàm tạo có đối để khởi gán tên
  {
    name= strdup(name1);
  }
  ~Animal()                   // hàm hủy
  {
    delete name;
  }
  void WhoAmI()
  {
    cout<<"An Animal";        // ☺
  }
};
```

Lớp Cat và Dog

```
class Cat: public Animal {
public:
  Cat(): Animal() { }
  Cat(char* name1): Animal(name1) { }
  void WhoAmI()
  {
    cout << "I am a Cat: " << name;
  }
};
```

```
class Dog: public Animal {
public:
  Dog(): Animal() { }
  Dog(char* name1): Animal(name1) { }
  void WhoAmI()
  {
    cout << "I am a Dog: " << name;
  }
};
```

Lớp Kennel

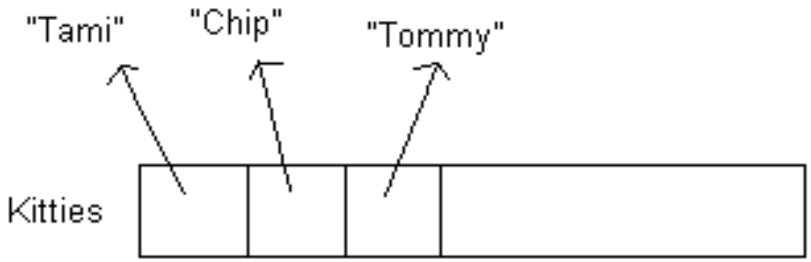
```
class Kennel
{
  unsigned int maxCat;
  unsigned int numCat;
  Cat** Kitties;

  unsigned int maxDog;
  unsigned int numDog;
  Dog** Doggies;

public:




};
```



Hàm tạo, hàm hủy

```
Kennel::Kennel(unsigned int maxc, unsigned int maxd)
{
  maxCat = maxc;
  maxDog = maxd;
  numCat = 0;
  numDog = 0;
  Kitties = new Cat*[maxCat];
  Doggies = new Dog*[maxDog];
  for(int i = 0; i < maxCat; ++i)
    Kitties[i] = NULL;
  for(int i = 0; i < maxDog; ++i)
    Doggies[i] = NULL;
}
```

```
Kennel::~Kennel()
{
  delete Kitties;
  delete Doggies;
}
```

Các phương thức Accept

```
unsigned int Kennel::Accept(Dog* d) {
  if(numDog == maxDog) return 0;
  ++numDog;
  int i= 0;
  while(Doggies[i] != NULL)
    i++;
  Doggies[i] = d;
  return (i+1);        // số hiệu ngăn vừa nhốt con vật thêm vào
}
```

Các phương thức Release

```
Dog* Kennel::ReleaseDog(unsigned int pen) {
  if(pen>maxDog) return NULL;
  --pen;
  if(Doggies[pen]!= NULL) {
    Dog* temp= Doggies[pen];  Doggies[pen]= NULL; --numDog;
    return temp;
  }
  else return NULL;
}
```

Phương thức ListAnimal

```
void ListAnimal()
{
  if(numDog>0)
    for(int i=0; i<maxDog; ++i)
     if(Doggies[i]!= NULL)
       {
          cout<<"\nThe dog in pen "<<i<<" said:";
          Doggies[i]->WhoAmI();
       }
  if(numCat>0)
    for(int i=0; i<maxCat; ++i)
     if(Kitties[i]!= NULL)
       {
          cout<<"\nThe cat in pen "<<i<<" said:";
          Kitties[i]->WhoAmI();
       }
}
```

Main

```
Dog d1("Milu");
Dog d2("Kaka");
Dog d3("John");
Cat c1("Tami");
Cat c2("Chip");
Cat c3("Tommy");
Cat c4("Nick");
void main() {
  Kennel k(10,10); // Max: 10 dogs, 10 cats
  k.Accept(&d1);
  unsigned int c2pen= k.Accept(&c2);
  k.Accept(&d2);
  k.Accept(&c1);
  unsigned int d3pen= k.Accept(&d3);
  k.Accept(&c4);
  k.ReleaseCat(c2pen);
  k.ReleaseDog(d3pen);
  k.Accept(&c3);
  k.ListAnimal();
}
```

```
The dog in pen 1 said: I am a dog Milu
The dog in pen 2 said: I am a dog Kaka
The cat in pen 1 said: I am a cat Tommy
The cat in pen 2 said: I am a cat Tami
The cat in pen 1 said: I am a cat Nick
```

Nhận xét

- Chương trình hoạt động được, nhưng còn một số hạn chế?
- **Hạn chế 1:** Số hiệu ngăn khi hiện ra không đúng
- **Hạn chế 2:** Lặp lại nhiều nội dung trong phương thức Accept
- **Hạn chế 3:** Tuy cùng Kennel nhưng quản lý chó và mèo riêng
  - Kennel k(10,10) → quản lý được tối đa 10 con chó và 10 con mèo
  - Nếu Suzy cần quản lý 15 con chó và 5 con mèo thì làm thế nào?
- **Hạn chế 4:** Nếu Suzy muốn nuôi thêm một loại con vật khác?
  - Thiết lập các thành phần dữ liệu để kiểm soát mảng con vật đó, ví dụ con lợn (Piggies)
  - Thay đổi các hàm tạo Kennel cho thích hợp
  - Thêm vào các phương thức mới trong Kennel để nhận một con lợn và lấy một con lợn ra khỏi ngăn
  - Điều chỉnh ListAnimal để hiện được cả các con lợn

Giải pháp

- Mục đích
  - Mỗi con vật nhốt vào một ngăn không phân biệt loại con vật
  - Các thao tác thêm và bớt con vật khỏi chuồng không cần quan tâm là con vật gì
  - Chỉ có việc "kêu" là khác nhau
- Cài đặt cụ thể?
  - Trong lớp Kennel: các phương thưc Accept và Release áp dụng chung cho Animal thay vì từng loại con vật
  - Trong lớp Animal: WhoAmI cần dùng phương thức ảo (virtual method)

Lớp Animal (cải tiến)

```
class Animal
{
private:
  char* name;
public:
  Animal() { name= NULL; }
  Animal(char* name1) { name= strdup(name1); }
  ~Animal()
  {
    delete name;
  }
  virtual void WhoAmI() = 0; // virtual method
};
```

Lớp Cat và Dog

```
class Cat: public Animal {
public:
  Cat(): Animal() { }
  Cat(char* name1): Animal(name1) { }
  void WhoAmI()
  {
    cout<<"I am a cat: "<<name;
  }
};
```

```
class Dog: public Animal {
public:
  Dog(): Animal() { }
  Dog(char* name1): Animal(name1) { }
  void WhoAmI()
  {
    cout<<"I am a dog: "<<name;
  }
};
```

Lớp Kennel (cải tiến)

```
class Kennel
{
  unsigned int maxAnimal;
  unsigned int numAnimal;
  Animal** residents;
public:
  Kennel(unsigned int max);
  ~Kennel();
  unsigned int Accept(Animal* a);    // nhận một con vật vào chuồng
  Animal* Release(unsigned int pen); // loại một con khỏi ngăn pen
  void ListAnimal();
};
```

Hàm tạo, hàm hủy

```
Kennel::Kennel(unsigned int max)
{
  maxAnimal= max;
  numAnimal= 0;
  residents= new Animal*[max];
  for(int i=0; i<max; ++i)
    residents[i]= NULL;
}
```

```
Kennel::~Kennel()
{
  delete residents;
}
```

Các phương thức Accept, Release

```
unsigned int Kennel::Accept(Animal* a){       // Xử lý chung!
  if(numAnimal==maxAnimal) return 0;
  ++numAnimal;
  int i= 0;
  while(residents[i]!= NULL)
    i++;
  residents[i]= a;
  return (i+1);  // ngăn thứ mấy, vì tính từ 0 nêu phải +1
}
```

```
Animal* Kennel::Release(unsigned int pen){    // Xử lý chung!
  if(pen>maxAnimal) return NULL;
  --pen;          // vì tính từ 0, nên phải -1
  if(residents[pen]!= NULL) {
    Animal* temp = residents[pen];
    residents[pen]= NULL; --numAnimal;
    return temp;
  }
  else return NULL;
}
```

Phương thức ListAnimal

```
void ListAnimal()
{
  if(numAnimal>0)
    for(int i=0; i<maxAnimal; ++i)
     if(residents[i]!= NULL)
       {
         cout<<"The animal in pen "<<i<<" said:";
         residents[i]->WhoAmI();
       }
}
```

Main (cải tiến)

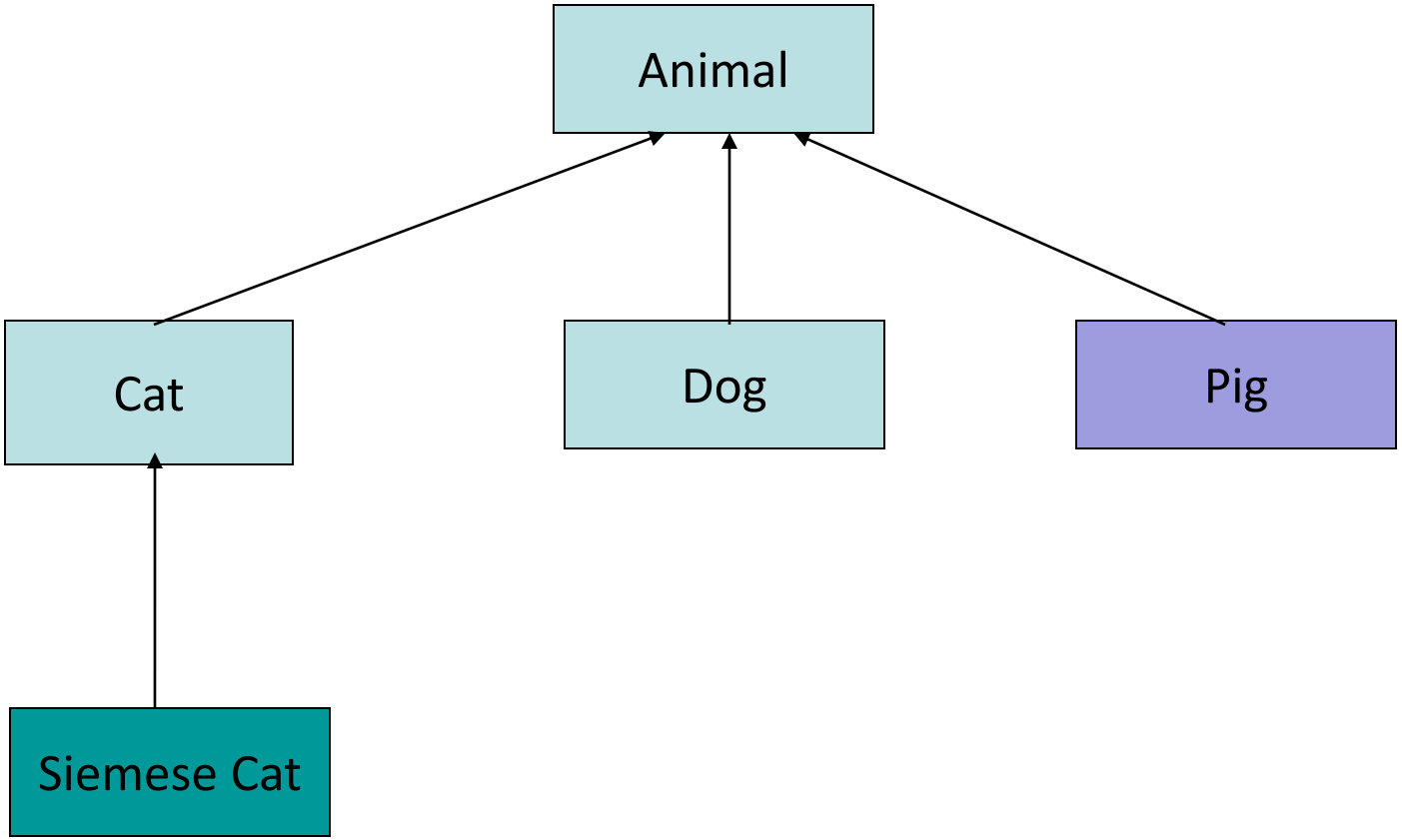
```
Dog d1("Milu");
Dog d2("Kaka");
Dog d3("John");
Cat c1("Tami");
Cat c2("Chip");
Cat c3("Tommy");
Cat c4("Nick");
void main() {
  Kennel k(20);       // Max: 20 animals
  k.Accept(&d1);
  unsigned int c2pen= k.Accept(&c2);
  k.Accept(&d2);
  k.Accept(&c1);
  unsigned int d3pen= k.Accept(&d3);
  k.Accept(&c4);
  k.Release(c2pen);
  k.Release(d3pen);
  k.Accept(&c3);
  k.ListAnimal();
}
```

```
The animal in pen 1 said: I am a dog Milu
The animal in pen 2 said: I am a cat Tommy
The animal in pen 3 said: I am a dog Kaka
The animal in pen 4 said: I am a cat Tami
The animal in pen 6 said: I am a cat Nick
```

Nhận xét về kết quả

- Kết quả giống như chương trình ban đầu
- Ngắn gọn và rõ ràng hơn!
- Khắc phục Hạn chế 1: Số con vật từng loại linh hoạt
  - Nếu chuồng có 20 ngăn có thể nhốt 10 chó & 10 mèo hoặc 15 chó & 5 mèo...
- Khắc phục Hạn chế 2
  - Dễ dàng mở rộng chương trình do các thao tác được xử lý theo cách chung
  - Các thao tác có đặc điểm riêng (Cách xưng tên của các con vật) được định nghĩa là hàm ảo

Mở rộng chương trình



Thêm các loại con vật mới

- Thêm các con lợn

```
class Pig: public Animal
{
public:
  Pig(): Animal();
  Pig(char* n): Animal(n);
  void WhoAmI() { cout<<"I am a pig: " <<name; }
};
```

- Thêm các con mèo Xiêm

```
class Siemese: public Cat
{
public:
  Siemese(): Cat();
  Siemese(char* n): Cat(n);
  void WhoAmI() { cout<<"I am a Siem cat: " <<name; }
};
```

Question

Object technology is . . . ?

A. A set of principles guiding software construction.
B. A new theory striving to gain acceptance.
C. A dynamic new language by Grady Booch.
D. Based on the principles of abstraction and modularity.

Answer: A

Question

A model . . .?

A. Is not necessary when team members understand their job.
B. Has to be structural AND behavioral.
C. Is a simplification of reality.
D. Is an excuse for building an elaborate plan.

Answer: C

Question

Why do we model?

A. Helps to visualize a system
B. Gives us a template for constructing a system
C. Documents our decisions
D. All of the above

Answer: D

Question

The best models are connected to . . .?

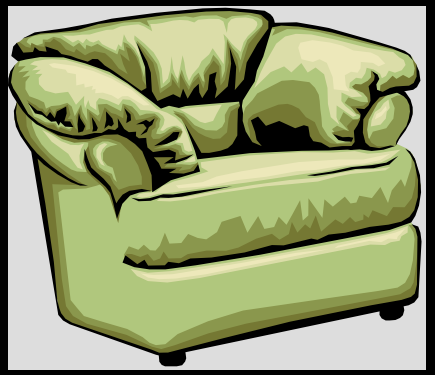
A. Java-script code
B. Reality
C. C ++
D. Issues that tie it to an object-oriented developer

Answer: B

Question

Which project would be least likely to require a model?

A. 

B. 

C. 

D. 

Answer: B

Question

Which principles of modeling are correct?

A. The model you create, influences how the problem is attacked.
B. The best kinds of models are those that let you chose your degree of detail.
C. The best models are connected to reality.
D. Create models that are built and studied separately.

Answer: A, B, C and D

Question

Encapsulation . . . ?

A. Allows direct manipulation of things that have been encapsulated.
B. Is often referred to as information hiding.
C. Causes costly and extensive maintenance.
D. Causes changes to affect clients during implementation.

Answer: B

Question

What happens when you incorporate modularity into your plan?

A. It reduces something complex into manageable pieces.
B. It builds modules that talk to each other.
C. Creates systems too large to understand.
D. Parts of your system cannot be independently developed.

Answer: A

Question

A class . . . ?

A. Is an encapsulation of an object.
B. Represents the hierarchy of an object.
C. Is an instance of an object.
D. Is an abstract definition of an object.

Answer: D

Question

Polymorphism can be described as?

A. Hiding many different implementations behind one interface
B. Inheritance
C. Information placing
D. Generalization

Answer: A