LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Chương 6: Kế thừa (p2)

Trình bày: ThS. Lê Thanh Trọng

NỘI DUNG

1. Phạm vi truy xuất trong kế thừa
2. Truy cập theo chiều dọc
3. Truy cập theo chiều ngang
4. Phương thức thiết lập, hủy bỏ trong kế thừa
5. Định nghĩa thành phần riêng
6. Đa kế thừa

NỘI DUNG

1. **Phạm vi truy xuất trong kế thừa**
2. Truy cập theo chiều dọc
3. Truy cập theo chiều ngang
4. Phương thức thiết lập, hủy bỏ trong kế thừa
5. Định nghĩa thành phần riêng
6. Đa kế thừa

Phạm vi truy xuất

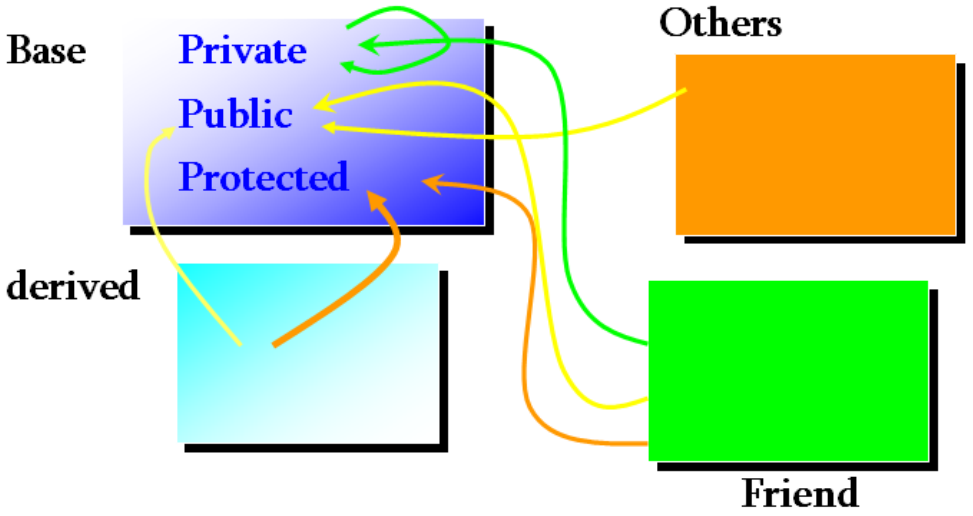
- Khi thiết lập quan hệ kế thừa, ta vẫn phải quan tâm đến tính đóng gói và che dấu thông tin
- Điều này ảnh hưởng đến phạm vi truy xuất của các thành phần của lớp

Phạm vi truy xuất

- Thuộc tính public:
  - Thành phần nào có thuộc tính public thì có thể truy xuất từ bất cứ nơi nào.
- Thuộc tính private: Thành phần có thuộc tính private
  - Là riêng tư của lớp đó
  - Chỉ có hàm thành phần của lớp và ngoại lệ các hàm bạn, lớp bạn được phép truy xuất.
  - Các lớp con không có quyền truy xuất

Phạm vi truy xuất

- Thuộc tính protected:
  - Cho phép qui định một vài thành phần nào đó của lớp là bảo mật, theo nghĩa thế giới bên ngoài không được phép truy xuất, nhưng tất cả các lớp con, cháu… đều được phép truy xuất.



Ví dụ thuộc tính private

```
class Nguoi {
    char *HoTen;
    int NamSinh;
public:
    //...
};
class SinhVien : public Nguoi {
    char *MaSo;
public:
    //...
    void Xuat() const;
};
```

Thuộc tính private

- Trong ví dụ trên, không có hàm thành phần nào của lớp SinhVien có thể truy xuất các thành phần HoTen, NamSinh của lớp Nguoi.
- Ví dụ, đoạn chương trình sau đây sẽ gây ra lỗi:

```
void SinhVien::Xuat() const {
        cout << "Sinh vien, ma so: “ << MaSo << ", ho ten:“ << HoTen;
}
```

Thuộc tính private

❖Ta có thể khắc phục lỗi trên nhờ khai báo lớp SinhVien là bạn của lớp Nguoi như trong ví dụ ban đầu:

```
class Nguoi {
    friend class SinhVien;
    char *HoTen;
    int NamSinh;
public:
    //...
};
```

Thuộc tính private

- Khai báo lớp bạn như trên, lớp SinhVien có thể truy xuất các thành phần private của lớp Nguoi.
- Cách làm trên chỉ giải quyết được nhu cầu của người sử dụng khi muốn tạo lớp con có quyền truy xuất các thành phần dữ liệu private của lớp cha.
- Tuy nhiên, cần phải sửa lại lớp cha và tất cả các lớp ở cấp cao hơn mỗi khi có một lớp con mới.

Thuộc tính private

```
class Nguoi {
    friend class SinhVien;
    friend class NuSinh;
    char *HoTen;  int NamSinh;
public:
    //...
    void An() const { cout << HoTen << " an 3 chen com";}
};
class SinhVien : public Nguoi {
    friend class NuSinh;
    char *MaSo;
public:
    //...
};
```

Thuộc tính protected

❖Trong ví dụ trước, khi cài đặt lớp NuSinh ta phải thay đổi lớp cha SinhVien và cả lớp cơ sở Nguoi ở mức cao hơn

```
class Nguoi {
    protected:
        char *HoTen;
        int NamSinh;
    public:
        //...
};
```

Thuộc tính protected

```
class SinhVien : public Nguoi {
protected:
    char *MaSo;
public:
    SinhVien(char *ht, char *ms, int ns) : Nguoi(ht,ns){
            MaSo = strdup(ms);
    }
    ~SinhVien(){
            delete [ ] MaSo;
    }
    void Xuat() const;
};
```

Thuộc tính protected

```
class NuSinh : public SinhVien {
public:
    NuSinh(char *ht, char *ms, int ns) : SinhVien(ht,ms,ns){
    }
    void An() const {
            cout << HoTen << " ma so " << MaSo << " an 2 to pho";
    }
};
// Co the truy xuat Nguoi::HoTen va
// Nguoi::NamSinh va SinhVien::MaSo
```

Thuộc tính protected

```
void Nguoi::Xuat() const {
    cout << "Nguoi, ho ten: " << HoTen << " sinh " << NamSinh;
}
void SinhVien::Xuat() const {
    cout << "Sinh vien, ma so: " << MaSo << ", ho ten: " << HoTen;
    // Ok: co quyen truy xuat, Nguoi::HoTen, Nguoi::NamSinh
}
void SinhVien::Xuat() const {
    cout << "Sinh vien, ma so: " << MaSo
    cout  << ", ho ten: " << HoTen;
}
```

Thuộc tính protected

- Là cách để tránh phải sửa đổi lớp cơ sở khi có lớp con mới hình thành → Đảm bảo tính đóng gói
- Thông thường ta dùng thuộc tính protected cho thành phần dữ liệu và public cho thành phần phương thức
- Tóm tại, thành phần có thuộc tính protected chỉ cho phép những lớp con kế thừa được phép sử dụng

Phạm vi truy xuất

- Truy xuất theo chiều dọc:
  - Hàm thành phần của lớp con có quyền truy xuất các thành phần của lớp cha hay không?
- Truy xuất theo chiều ngang:
  - Các thành phần của lớp cha, sau khi kế thừa xuống lớp con, thì thế giới bên ngoài có quyền truy xuất thông qua đối tượng của lớp con hay không?

NỘI DUNG

1. Phạm vi truy xuất trong kế thừa
2. **Truy cập theo chiều dọc**
3. Truy cập theo chiều ngang
4. Phương thức thiết lập, hủy bỏ trong kế thừa
5. Định nghĩa thành phần riêng
6. Đa kế thừa

Truy xuất theo chiều dọc

- Lớp con có quyền truy xuất các thành phần của lớp cha hay không, hoàn toàn do lớp cha quyết định. Điều đó được xác định bằng **phạm vi truy cập**
- Trong trường hợp lớp Sinh viên kế thừa lớp Người, Sinh viên có quyền truy xuất họ tên của chính mình (được khai báo ở lớp Người) hay không?

Truy xuất theo chiều dọc

```
class A{
private:
    int a;
    void f();
protected:
    int b;
    void g();
public:
    int c;
    void h();
};
```

```
void A::f()
{
    a = 1;          b = 2;          c = 3;
}
void A::g()
{
    a = 4;          b = 5;          c = 6;
}
void A::h(){
    a = 7;          b = 8;          c = 9;
}
```

Truy xuất theo chiều dọc

❖ Ví dụ: Cho biết trong đoạn chương trình sau câu lệnh nào đúng, câu lệnh nào sai.

```
void main()
{
    A x;
    x.a = 10;
    x.f();
    x.b = 20;
    x.g();
    x.c = 30;
    x.h();
}
```

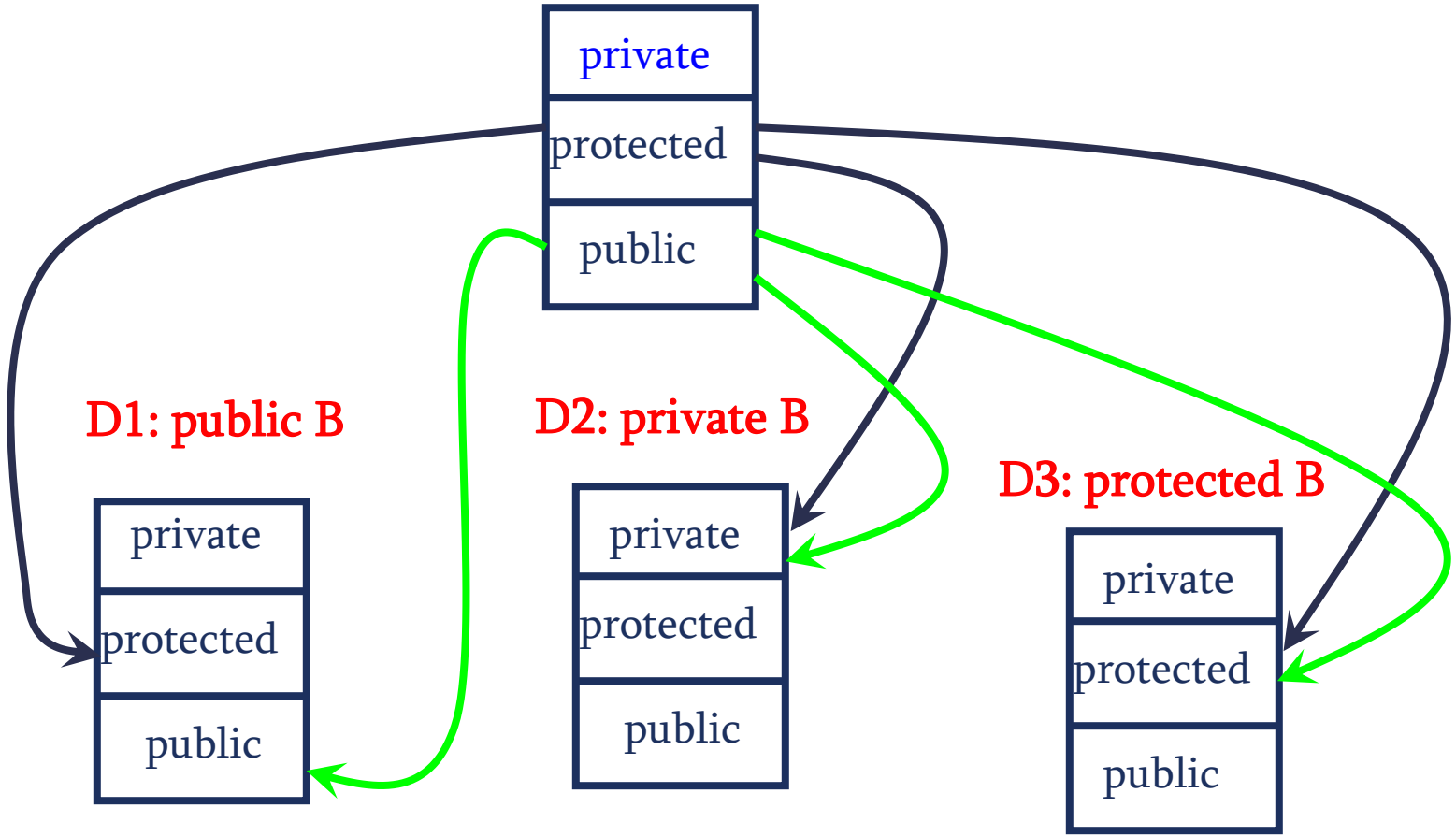
NỘI DUNG

1. Phạm vi truy xuất trong kế thừa
2. Truy cập theo chiều dọc
3. **Truy cập theo chiều ngang**
4. Phương thức thiết lập, hủy bỏ trong kế thừa
5. Định nghĩa thành phần riêng
6. Đa kế thừa

Truy xuất theo chiều ngang

- Thành phần protected và public của lớp khi đã kế thừa xuống lớp con thì thế giới bên ngoài có quyền truy xuất thông qua đối tượng thuộc lớp con hay không?
- Điều này hoàn toàn do lớp con quyết định bằng phạm vi kế thừa: Kế thừa public, Kế thừa protected, Kế thừa private

Phạm vi truy xuất trong kế thừa



Phạm vi truy xuất trong kế thừa

Phạm vi kế thừa

Phạm vi truy cập	private	protected	public
private	x	x	x
protected	private	protected	protected
public	private	protected	public

NỘI DUNG

1. Phạm vi truy xuất trong kế thừa
2. Truy cập theo chiều dọc
3. Truy cập theo chiều ngang
4. **Phương thức thiết lập, hủy bỏ trong kế thừa**
5. Định nghĩa thành phần riêng
6. Đa kế thừa

Phương thức thiết lập

- Phương thức thiết lập của lớp cơ sở luôn luôn được gọi (được tạo trước) mỗi khi có một đối tượng của lớp dẫn xuất được tạo ra (được tạo sau)

- Nếu mọi phương thức thiết lập của lớp cơ sở đều đòi hỏi phải cung cấp tham số thì lớp con bắt buộc phải có phương thức thiết lập để cung cấp các tham số đó

Phương thức thiết lập

❖Ví dụ 1:

```
class A {
  public:
    A ( )
     {  cout<< “A:default”<<endl;  }
    A (int a){
         cout<<“A:parameter”<<endl;
    }
};
```

```
class B : public A {
  public:
    B (int a){
       cout<<“B”<<endl;
    }
};
```

```
B test(1);
```

output:

```
A:default
B
```

Phương thức thiết lập

❖Ví dụ 2:

```
class A {
  public:
    A ( )
     {  cout<< “A:default”<<endl; }
    A (int a){
      cout<<“A:parameter”<<endl;
    }
};
```

```
class C : public A
{
  public:
    C (int a) : A(a){
        cout<<“C”<<endl;
    }
};
```

```
C test(1);
```

output:

```
A:parameter
C
```

Phương thức hủy bỏ

- Khi một đối tượng bị hủy đi, phương thức hủy bỏ của nó sẽ được gọi (trước), sau đó, các phương thức hủy bỏ của lớp cơ sở sẽ được gọi một cách tự động (sau)
- Vì vậy, lớp con không cần và cũng không được thực hiện các thao tác dọn dẹp cho các thành phần thuộc lớp cha

Phương thức hủy bỏ - Ví dụ

```
class SinhVien : public Nguoi {
    char *MaSo;
public:
    SinhVien( char *ht, char *ms, int ns) : Nguoi(ht,ns) {
        MaSo = strdup(ms);
    }
    SinhVien(const SinhVien &s) : Nguoi(s){
        MaSo = strdup(s.MaSo);
    }
    ~SinhVien() {delete [ ] MaSo;}
    //...
};
```

NỘI DUNG

1. Phạm vi truy xuất trong kế thừa
2. Truy cập theo chiều dọc
3. Truy cập theo chiều ngang
4. Phương thức thiết lập, hủy bỏ trong kế thừa
5. **Định nghĩa thành phần riêng**
6. Đa kế thừa

Định nghĩa các thành phần riêng

❖Ngoài các thành phần được kế thừa, lớp dẫn xuất có thể định nghĩa thêm các thành phần riêng

```
class HinhTron : Diem {
    double r;
public:
    HinhTron( double tx, double ty, double rr) : Diem(tx, ty){
            r = rr;
    }
    void Ve(int color) const;
    void TinhTien( double dx, double dy) const;
};
HinhTron t(200,200,50);
```

Định nghĩa các thành phần riêng

❖Lớp dẫn xuất cũng có thể override các phương thức đã được định nghĩa ở trong lớp cha.

```
class A {
  protected:
     int x, y;
   public:
     void print (){
        cout<<“From A”<<endl;
     }
};
```

```
class B : public A
{
   public:
     void print (){
        cout<<“From B”<<endl;
     }
};
```

Truy cập phương thức

```
class Point{
    protected:
      int x, y;
    public:
      void set(int a, int b)
           { x=a; y=b; }
      void foo ();
      void print();
};
```

```
Point A;
A.set(30,50);   ???
A.print();
```

```
class Circle : public Point{
  private:  double r;
  public:
     void set (int a, int b, double c) {
          Point ::set(a, b); //cùng tên hàm gọi đến
          r = c;
     }
     void print() { //.. }
};
```

```
     Circle C;
     C.set(10,10,100);   ???
     C.foo ();           ???
     C.print();          ???
```

Con trỏ và kế thừa

- Con trỏ trong kế thừa hoạt động theo nguyên tắc sau:
  - Con trỏ trỏ đến đối tượng thuộc lớp cơ sở thì **có thể** trỏ đến các đối tượng thuộc lớp con
  - Nhưng con trỏ trỏ đến đối tượng thuộc lớp con thì **không thể** trỏ đến các đối tượng thuộc lớp cơ sở
  - Có thể ép kiểu để con trỏ trỏ đến đối tượng thuộc lớp con có thể trỏ đến đối tượng thuộc lớp cơ sở. Tuy nhiên thao tác này có thể nguy hiểm

NỘI DUNG

1. Phạm vi truy xuất trong kế thừa
2. Truy cập theo chiều dọc
3. Truy cập theo chiều ngang
4. Phương thức thiết lập, hủy bỏ trong kế thừa
5. Định nghĩa thành phần riêng
6. **Đa kế thừa**

Đa kế thừa

❖Đa kế thừa cho phép một lớp có thể là dẫn xuất của nhiều lớp cơ sở

```
class A : public B, public C {
        …
};
```

❖Các đặc điểm của kế thừa đơn vẫn đúng cho trường hợp đa kế thừa

❖Ví dụ

- ❑Thủy phi cơ vừa có thể bay (máy bay), vừa có thể di chuyển trên nước (tàu)
- ❑Flycam vừa có thể bay (máy bay), vừa có thể chụp ảnh (máy ảnh)

Đa kế thừa – Ví dụ

```
class BASE_A
{
    public:
        int a;
        int f( ){
                return 0;
        }
        int g( ){
                return 0;
        }
        int h( ) { return 0;}
};
```

```
class BASE_B
{
    public:
        int a;
        int f( ){
                return 0;
        }
        int g( ){
                return 0;
        }
};
```

Đa kế thừa – Ví dụ

```
class ClassC : public BASE_A, public BASE_B{
    //…
};

void main(){
    ClassC C;
    C.BASE_A::f();
    C.BASE_A::a = 1;
    C.BASE_B::g();
    C.h();
}
```

Tóm tắt bài học kế thừa (p2)

- **Truy cập theo chiều dọc:** Lớp con chỉ có thể truy cập các thành phần public và protected ở lớp cha
- Truy cập theo chiều ngang:
  - Thành phần protected và public của lớp khi đã kế thừa xuống lớp con thì thế giới bên ngoài có quyền truy xuất thông qua đối tượng thuộc lớp con hay không?
  - Điều này hoàn toàn do lớp con quyết định bằng phạm vi kế thừa: kế thừa public, kế thừa protected, kế thừa private

Phạm vi truy cập \ Phạm vi kế thừa	private	protected	public
private	x	x	x
protected	private	protected	protected
public	private	protected	public

Tóm tắt bài học kế thừa (p2)

- Phương thức thiết lập của lớp cơ sở luôn luôn được gọi (được tạo trước) mỗi khi có một đối tượng của lớp dẫn xuất được tạo ra (được tạo sau)
- Khi một đối tượng bị hủy đi, phương thức hủy bỏ của nó sẽ được gọi (trước), sau đó, các phương thức hủy bỏ của lớp cơ sở sẽ được gọi một cách tự động (sau), lớp con không cần và cũng không được thực hiện các thao tác dọn dẹp cho các thành phần thuộc lớp cha

Tóm tắt bài học kế thừa (p2)

- Ngoài các thành phần được kế thừa, lớp dẫn xuất có thể định nghĩa thêm các thành phần riêng
- Lớp dẫn xuất cũng có thể override các phương thức đã được định nghĩa ở trong lớp cha
- Con trỏ trỏ đến đối tượng thuộc lớp cơ sở thì có thể trỏ đến các đối tượng thuộc lớp con, nhưng con trỏ trỏ đến đối tượng thuộc lớp con thì không thể trỏ đến các đối tượng thuộc lớp cơ sở

Bài tập

1. Xây dựng các loại đối tượng sinh viên (mã sinh viên, điểm trung bình, xếp loại), công nhân (lương cơ bản, số năm kinh nghiệm), ca sĩ (hạng (A, B, C, D), cát sê) và có các thông tin chung gồm: họ tên, ngày tháng năm sinh, số CCCD. Viết chương trình cho phép nhập vào một trong các loại đối tượng kể trên. In thông tin đối tượng đó.
2. Tạo một danh sách các đối tượng, mỗi đối tượng thuộc một trong các loại: sinh viên, học sinh, công nhân, nghệ sĩ, ca sĩ. Viết chương trình cho phép nhập danh sách kể trên, in thông tin của từng đối tượng trong danh sách.
3. Xây dựng lớp biểu diễn khái niệm hình ellipse và hình tròn (ellipse có 2 bán kính bằng nhau). Viết chương trình cho phép nhập vào một hình (tròn hoặc eclipse). Xuất thông tin hình đó.

Bài tập

1. Xây dựng các loại đối tượng sinh viên (mã sinh viên, điểm trung bình, xếp loại), công nhân (lương cơ bản, số năm kinh nghiệm), ca sĩ (hạng (A, B, C, D), cát sê) và có các thông tin chung gồm: họ tên, ngày tháng năm sinh, số CCCD. Viết chương trình cho phép nhập vào một trong các loại đối tượng kể trên. In thông tin đối tượng đó.
2. Tạo một danh sách các đối tượng, mỗi đối tượng thuộc một trong các loại: sinh viên, học sinh, công nhân, nghệ sĩ, ca sĩ. Viết chương trình cho phép nhập danh sách kể trên, in thông tin của từng đối tượng trong danh sách.
3. Xây dựng lớp biểu diễn khái niệm hình ellipse và hình tròn (ellipse có 2 bán kính bằng nhau). Viết chương trình cho phép nhập vào một hình (tròn hoặc eclipse). Xuất thông tin hình đó.

Bài tập

4. Xây dựng các loại đối tượng hình thang, hình bình hành, hình chữ nhật, hình vuông. Chỉ xét các hình thang, hình bình hành có đáy song song với trục hoành, chỉ xét hình chữ nhật và hình vuông có cạnh song song với trục toạ độ. Viết chương trình cho phép nhập vào danh sách các hình kể trên, tính chu vi, diện tích các hình và xuất ra thông tin chình đó.
5. Một nông trại chăn nuôi có 3 loại gia súc: bò, cừu, và dê. Mỗi loại gia súc đều có thể sinh con, cho sữa và phát ra tiếng kêu riêng của chúng. Khi đói, các gia súc sẽ phát ra tiếng kêu để đòi ăn. Sau một thời gian chăn nuôi, người chủ nông trại muốn thống kê xem trong nông trại có bao nhiêu gia súc ở mỗi loại, tổng số lit sữa mà tất cả các gia súc của ông đã cho.
   Áp dụng kế thừa, xây dựng chương trình cho phép người chủ nông trại nhập vào số lượng gia súc ban đầu ở mỗi loại.
   a. Một hôm người chủ nông trại đi vắng, tất cả gia súc trong nông trại đều đói. Hãy cho biết những tiếng kêu nghe được trong nông trại.
   b. Chương trình sẽ đưa ra thống kê các thông tin người chủ mong muốn (nêu trên) sau một lứa sinh và một lược cho sữa của tất cả gia súc. Biết rằng:
      - Tất cả gia súc ở mỗi loại đều sinh con.
      - Số lượng sinh của mỗi gia súc là ngẫu nhiên.
      - Tất cả gia súc ở mỗi loại đều cho sữa.
      - Số lit sữa mỗi gia súc cho là ngẫu nhiên nhưng trong giới hạn sau:
        - Bò: 0 – 20 lít.
        - Cừu: 0 – 5 lít.
        - Dê: 0 – 10 lít.