Πολλαπλή Κληρονομικότητα

Εργίνα Καβαλλιεράτου

Σημερινό Μάθημα

- ✓Πολλαπλή κληρονομικότητα
- ✓Constructors σε αντικείμενα πολλαπλής κληρονομικότητας
- ✓Διπλή συνάρτηση
- ✓Κοινή κλάση βάσης
- ✓Virtual Κληρονομικότητα
- ✓Mixin classes
- ✓Αφηρημένοι τύποι δεδομένων
- ✓Pure Virtual συναρτήσεις

Πολλαπλή Κληρονομικότητα

- ✓Ας υποθέσουμε ότι με τη βοήθεια της Ιεραρχίας έχουμε χωρίσει την κλάση των ζώων σε πουλιά και θηλαστικά.
- ✓Η κλάση των πουλιών, Bird, εμπεριέχει τη συνάρτηση Fly(), ενώ στα θηλαστικά ανήκει η κλάση Horse που περιέχει τις συναρτήσεις Whinny() και Gallop().
- ✓Έστω ότι χρειαζόμαστε την κλάση Pegasus που χρειάζεται τις συναρτήσεις Fly(), Whinny(), και Gallop(). Ποια πρέπει να είναι η βάση μας

Πολλαπλή Κληρονομικότητα

✓Είναι δυνατό να παράγουμε μία κλάση από δύο βάσεις:

class Pegasus : public Horse, public Bird

✓Αυτό ονομάζεται πολλαπλή κληρονομικότητα.

✓Τότε η παραγόμενη κληρονομεί και από τις δύο βάσεις

πολλαπλή κληρονομικότητα /1

```
#include <iostream.h>
   class Horse  {
    public:
      Horse() { cout << "Horse constructor... "; }
      virtual ~Horse() { cout << "Horse destructor... "; }
      virtual void Whinny() const { cout << "Whinny!... "; }
   private:
      int itsAge;  };
```

πολλαπλή κληρονομικότητα /2

```
class Bird   {
   public:
     Bird() { cout << "Bird constructor... "; }
     virtual ~Bird() { cout << "Bird destructor... "; }
     virtual void Chirp() const { cout << "Chirp... ";  }
     virtual void Fly() const {  cout << "I can fly! I can fly! I can fly! ";   }
   protected:
     int itsWeight;  };
```

πολλαπλή κληρονομικότητα /3

```
class Pegasus : public Horse, public Bird
{
      public:
       void Chirp() const { Whinny(); }
       Pegasus() { cout << "Pegasus constructor... "; }
       ~Pegasus() { cout << "Pegasus destructor... "; }
};
```

πολλαπλή κληρονομικότητα /4

```
const int MagicNumber = 2;
int main()   {
   Horse* Ranch[MagicNumber];
   Bird* Aviary[MagicNumber];
   Horse * pHorse;
   Bird * pBird;
   int choice,i;
```

πολλαπλή κληρονομικότητα /5

```
for (i=0; i<MagicNumber; i++)  {
   cout << "\n(1)Horse (2)Pegasus: ";
   cin >> choice;
   if (choice == 2)
      pHorse = new Pegasus;
   else
      pHorse = new Horse;
   Ranch[i] = pHorse;   }
```

πολλαπλή κληρονομικότητα /6

```
for (i=0; i<MagicNumber; i++)  {
      cout << "\n(1)Bird (2)Pegasus: ";
      cin >> choice;
      if (choice == 2)
        pBird = new Pegasus;
      else
        pBird = new Bird;
      Aviary[i] = pBird;
   }
```

πολλαπλή κληρονομικότητα /7

```
cout << "\n";
for (i=0; i<MagicNumber; i++){
        cout << "\nRanch[" << i << "]: " ;
        Ranch[i]->Whinny();
        delete Ranch[i];}
for (i=0; i<MagicNumber; i++){
        cout << "\nAviary[" << i << "]: " ;
        Aviary[i]->Chirp();
        Aviary[i]->Fly();
        delete Aviary[i];}
return 0;
}
```

πολλαπλή κληρονομικότητα /έξοδος

(1)Horse (2)Pegasus: 1
Horse constructor...
(1)Horse (2)Pegasus: 2
Horse constructor... Bird constructor... Pegasus constructor...
(1)Bird (2)Pegasus: 1
Bird constructor...
(1)Bird (2)Pegasus: 2
Horse constructor... Bird constructor... Pegasus constructor...
Ranch[0]: Whinny!... Horse destructor...
Ranch[1]: Whinny!... Pegasus destructor... Bird destructor... Horse destructor...
Aviary[0]: Chirp... I can fly! I can fly! I can fly! Bird destructor...
Aviary[1]: Whinny!... I can fly! I can fly! I can fly!
Pegasus destructor... Bird destructor... Horse destructor...

Constructors σε αντικείμενα με πολλαπλή κληρονομικότητα

- ✓Όταν μία κλάση έχει δύο κλάσεις βάσεις που έχουν δομητές με παραμέτρους πρέπει να τους αρχικοποιεί στη σειρά.

Πολλαπλοί constructor/1

```
#include <iostream.h>
   typedef int HANDS;
   enum COLOR { Red, Green, Blue, Yellow, White, Black, Brown } ;
   enum BOOL { FALSE, TRUE };
```

Πολλαπλοί constructor/2

```
class Horse   {
   public:
      Horse(COLOR color, HANDS height);
      virtual ~Horse() { cout << "Horse destructor...\n"; }
      virtual void Whinny()const { cout << "Whinny!... "; }
      virtual HANDS GetHeight() const { return itsHeight; }
      virtual COLOR GetColor() const { return itsColor; }
   protected:
      HANDS itsHeight;
      COLOR itsColor;  }
```

Πολλαπλοί constructor/3

```
class Bird  {
  public:
    Bird(COLOR color, BOOL migrates);
    virtual ~Bird() {cout << "Bird destructor...\n";  }
    virtual void Chirp()const { cout << "Chirp... ";  }
    virtual void Fly()const     {          cout << "I can fly! I can fly! I can fly! ";    }
    virtual COLOR GetColor()const { return itsColor; }
    virtual BOOL GetMigration() const { return itsMigration; }
  proteted:
    COLOR itsColor;       BOOL itsMigration;    };
```

Πολλαπλοί constructor/4

```
Horse::Horse(COLOR color, HANDS height): itsColor(color),itsHeight(height)
   {
      cout << "Horse constructor...\n";
   }

Bird::Bird(COLOR color, BOOL migrates):
      itsColor(color), itsMigration(migrates)
  {
      cout << "Bird constructor...\n";
   }
```

Πολλαπλοί constructor/5

```
class Pegasus : public Horse, public Bird    {
  public:
    void Chirp()const { Whinny(); }     Pegasus(COLOR, HANDS, BOOL,long);
    ~Pegasus() {cout << "Pegasus destructor...\n";}
    virtual long GetNumberBelievers() const   { return  itsNumberBelievers;    }
  private:
    long itsNumberBelievers;  };
  Pegasus::Pegasus(COLOR aColor,HANDS height,BOOL migrates, long NumBelieve):
 Horse(aColor, height),Bird(aColor, migrates),
    itsNumberBelievers(NumBelieve)  {  cout << "Pegasus constructor...\n“;  }
```

Πολλαπλοί constructor/6

```
int main()  {
    Pegasus *pPeg = new Pegasus(Red, 5, TRUE, 10);
    pPeg->Fly();        pPeg->Whinny();
    cout << "\nYour Pegasus is " << pPeg->GetHeight();
    cout << " hands tall and ";
    if (pPeg->GetMigration())
      cout << "it does migrate.";
    else
      cout << "it does not migrate.“
```

Πολλαπλοί constructor/7

```
  cout << "\nA total of " << pPeg->GetNumberBelievers();
  cout << " people believe it exists.\n";
  delete pPeg;
return 0; }
```

Πολλαπλοί constructor/έξοδος

Horse constructor...

Bird constructor...

Pegasus constructor...

I can fly! I can fly! I can fly! Whinny!...

Your Pegasus is 5 hands tall and it does migrate.

A total of 10 people believe it exists.

Pegasus destructor...

Bird destructor...

Horse destructor...

Διπλή συνάρτηση

- ✓Στην περίπτωση που μία συνάρτηση υπάρχει σε δύο κλάσεις βάσης μιας παραγόμενης κλάσης και κληθεί από την παραγόμενη κλάση, θα προκύψει σφάλμα.
- ✓Το πρόβλημα μπορεί να ξεπεραστεί με δύο τρόπους:
  - Είτε ορίζοντας ακριβώς ποια συνάρτηση καλούμε

    COLOR currentColor = pPeg->Horse::GetColor();
  - Είτε κάνοντας override τη συνάρτηση στην παραγόμενη κλάση

Κοινή κλάση βάσης/1

```
#include <iostream.h>
 typedef int HANDS;
enum COLOR { Red, Green, Blue, Yellow, White, Black, Brown } ;
enum BOOL { FALSE, TRUE };
class Animal  {
public:
  Animal(int);
  virtual ~Animal() { cout << "Animal destructor...\n"; }
  virtual int GetAge() const { return itsAge; }
  virtual void SetAge(int age) { itsAge = age; }
```

Κοινή κλάση βάσης/2

```
  private:
     int itsAge;    };
Animal::Animal(int age):  itsAge(age)   {
     cout << "Animal constructor...\n";   }
```

Κοινή κλάση βάσης/3

```
class Horse : public Animal   {
  public:
    Horse(COLOR color, HANDS height, int age);
    virtual ~Horse() { cout << "Horse destructor...\n"; }
    virtual void Whinny()const { cout << "Whinny!... "; }
    virtual HANDS GetHeight() const { return itsHeight; }
    virtual COLOR GetColor() const { return itsColor; }
  protected:
    HANDS itsHeight;
    COLOR itsColor    };
```

Κοινή κλάση βάσης/4

```
Horse::Horse(COLOR color, HANDS height, int age):
    Animal(age),   itsColor(color),itsHeight(height)    {
    cout << "Horse constructor...\n";   }
```

Κοινή κλάση βάσης/5

```
class Bird : public Animal    {
  public:
    Bird(COLOR color, BOOL migrates, int age);
    virtual ~Bird() {cout << "Bird destructor...\n";  }
    virtual void Chirp()const { cout << "Chirp... ";  }
    virtual void Fly()const
       { cout << "I can fly! I can fly! I can fly! "; }
    virtual COLOR GetColor()const { return itsColor; }
    virtual BOOL GetMigration() const { return itsMigration; }
  protected:
    COLOR itsColor;
    BOOL itsMigration;    };
```

Κοινή κλάση βάσης/6

```
Bird::Bird(COLOR color, BOOL migrates, int age):
    Animal(age),   itsColor(color), itsMigration(migrates)    {
    cout << "Bird constructor...\n";    }
```

Κοινή κλάση βάσης/7

```
class Pegasus : public Horse, public Bird    {
  public:
    void Chirp()const { Whinny(); }
    Pegasus(COLOR, HANDS, BOOL, long, int);
    ~Pegasus() {cout << "Pegasus destructor...\n";}
    virtual long GetNumberBelievers() const
      { return  itsNumberBelievers; }
    virtual COLOR GetColor()const { return Horse::itsColor; }
    virtual int GetAge() const { return Horse::GetAge(); }
  private:
    long itsNumberBelievers;   };
```

Κοινή κλάση βάσης/8

```
Pegasus::Pegasus(COLOR aColor, HANDS height, BOOL migrates,
    long NumBelieve, int age):   Horse(aColor, height,age), Bird(aColor,
  migrates,age),   itsNumberBelievers(NumBelieve) {
    cout << "Pegasus constructor...\n"; }
```

Κοινή κλάση βάσης/9

```
int main()
  {
    Pegasus *pPeg = new Pegasus(Red, 5, TRUE, 10, 2);
    int age = pPeg->GetAge();
    cout << "This pegasus is " << age << " years old.\n";
   delete pPeg;
  return 0;
}
```

Κοινή κλάση βάσης/έξοδος

Animal constructor...

Horse constructor...

Animal constructor...

Bird constructor...

Pegasus constructor...

This pegasus is 2 years old.

Pegasus destructor...

Bird destructor...

Animal destructor...

Horse destructor...

Animal destructor...

Virtual Κληρονομικότητα

- ✓Η χρήση μιας virtual κλάσης βάσης μας επιτρέπει τη χρήση κοινών συναρτήσεων ορισμένων ως virtual στη βάση, χωρίς να δημιουργεί πολλαπλά αντίγραφα.
- ✓Κανονικά, ο δομητής μιας κλάσης, αρχικοποιεί μόνο τις μεταβλητές της κλάσης του και τη βάση του.
- ✓Οι virtual κλάσεις βάσης αποτελούν εξαίρεση καθώς αρχικοποιούνται και από την τελευταία στη σειρά παραγόμενη κλάση.

Virtual Κληρονομικότητα/1

```
#include <iostream.h>
   typedef int HANDS;
   enum COLOR { Red, Green, Blue, Yellow, White, Black, Brown } ;
   enum BOOL { FALSE, TRUE };
   class Animal   {
    Animal(int);
    virtual ~Animal() { cout << "Animal destructor...\n"; }
    virtual int GetAge() const { return itsAge; }
    virtual void SetAge(int age) { itsAge = age; }
    int itsAge; };
  Animal::Animal(int age):   itsAge(age)    {
    cout << "Animal constructor...\n"; }
```

Virtual Κληρονομικότητα/2

```
class Horse : virtual public Animal    {
    Horse(COLOR color, HANDS height, int age);
    virtual ~Horse() { cout << "Horse destructor...\n"; }
    virtual void Whinny()const { cout << "Whinny!... "; }
    virtual HANDS GetHeight() const { return itsHeight; }
    virtual COLOR GetColor() const { return itsColor; }
  protected:
    HANDS itsHeight;
    COLOR itsColor;    };
  Horse::Horse(COLOR color, HANDS height, int age):
   Animal(age), itsColor(color),itsHeight(height)
  {      cout << "Horse constructor...\n";    }
```

Virtual Κληρονομικότητα/3

```
class Bird : virtual public Animal    {
    Bird(COLOR color, BOOL migrates, int age);
    virtual ~Bird() {cout << "Bird destructor...\n";  }
    virtual void Chirp()const { cout << "Chirp... ";  }
    virtual void Fly()const  { cout << "I can fly! I can fly! I can fly! "; }
    virtual COLOR GetColor()const { return itsColor; }
    virtual BOOL GetMigration() const { return itsMigration; }
    COLOR itsColor;
    BOOL itsMigration;  };
Bird::Bird(COLOR color, BOOL migrates, int age):
    Animal(age), itsColor(color), itsMigration(migrates)  {
    cout << "Bird constructor...\n";    }
```

Virtual Κληρονομικότητα/4

```
class Pegasus : public Horse, public Bird   {
    void Chirp()const { Whinny(); }
    Pegasus(COLOR, HANDS, BOOL, long, int);
    ~Pegasus() {cout << "Pegasus destructor...\n";}
    virtual long GetNumberBelievers() const
      { return  itsNumberBelievers; }
    virtual COLOR GetColor()const { return Horse::itsColor; }
    long itsNumberBelievers;   };
 Pegasus::Pegasus(COLOR aColor,HANDS height,BOOL migrates, long NumBelieve, int age):
 Horse(aColor, height,age), Bird(aColor, migrates,age), Animal(age*2),
itsNumberBelievers(NumBelieve)
{    cout << "Pegasus constructor...\n";    }
```

Friend class

- ✓Εάν θέλετε να εκθέσετε τα private δεδομένα μελών ή συναρτήσεις σε άλλη κλάση, πρέπει να τη δηλώσετε Friend κλάση.
- ✓Αυτό επεκτείνει την κλάση σας και συμπεριλάβει την Friend κλάση.
- ✓Είναι σημαντικό να σημειωθεί ότι η σχέση Friend δεν μπορεί να μεταφερθεί.
- ✓Ούτε κληρονομείται η σχέση Friend.
- ✓Τέλος, δεν είναι ανταλλακτική

Παράδειγμα 1

```
#include <iostream>
class Filos {
    int i =3;
    friend class Ektipwsi; };
class Ektipwsi {
  public:
    void display(Filos &a)     {
        cout<<"The value of i is : "<<a.i;  }
};
```

Παράδειγμα 2

```
int main(){
    Filos a;
    Ektipwsi b;
    b.display(a);
    return 0;
}
```

```
The value of i is : 3
```

Friend συναρτήσεις

- ✓Μερικές φορές θα θέλετε να παραχωρήσετε αυτό το επίπεδο πρόσβασης όχι σε μια ολόκληρη class, αλλά μόνο σε μία ή δύο συναρτήσεις της κλάσης
- ✓Μπορείτε να το κάνετε αυτό δηλώνοντας ότι είναι οι συναρτήσεις Friend μέλη.
- ✓Στην πραγματικότητα, μπορείτε να δηλώσετε οποιαδήποτε συνάρτηση ως Friend.

Παράδειγμα 1

```
#include <iostream>
class clasA{
    private:
    int A_value;
    public:
    clasA() {A_value = 20;}
    friend class clasB;  };
class clasB{
    private:
    int B_value;
    public:
```

Παράδειγμα 2

```
    void display(clasA& i) {
        cout<<"The private member's value accessed using friend class is: " << i.A_value<<endl; }
};

int main(){
cout<<"Welcome!"<<endl<<endl;
clasA A_value;
clasB B_value;
B_value.display(A_value);
return 0;
}
```

```
Welcome!

The private member's value accessed using friend class is: 20
```