Πανεπιστήμιο Αιγαίου

Μεθοδολογίες και Γλώσσες Προγραμματισμού Ι

Πολλαπλή Κληρονομικότητα

Εργίνα Καβαλλιεράτου (kavallieratou@aegean.gr)

Μόνιμη Επίκουρος Καθηγήτρια

Τμήμα Μηχανικών Πληροφοριακών & Επικοινωνιακών Συστημάτων

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αιγαίου**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σήμερα!

- ✓ Ανάγκη για πολλαπλή κληρονομικότητα
- ✓ Πολλαπλή κληρονομικότητα
- ✓ Constructors σε αντικείμενα πολλαπλής κληρονομικότητας
- ✓ Διπλή συνάρτηση
- ✓ Κοινή κλάση βάσης

Ανάγκη για 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;  };
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 /2

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

πολλαπλή κληρονομικότητα /3

```
const int MagicNumber = 2;
int main()   {
   Horse* Ranch[MagicNumber];
   Bird* Aviary[MagicNumber];
   Horse * pHorse;
   Bird * pBird;
   int choice,i;
   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 /4

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

πολλαπλή κληρονομικότητα /5

```
    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 };
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/2

```
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/3

```
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/3

```
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/4

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

Πολλαπλοί constructor/έξοδος

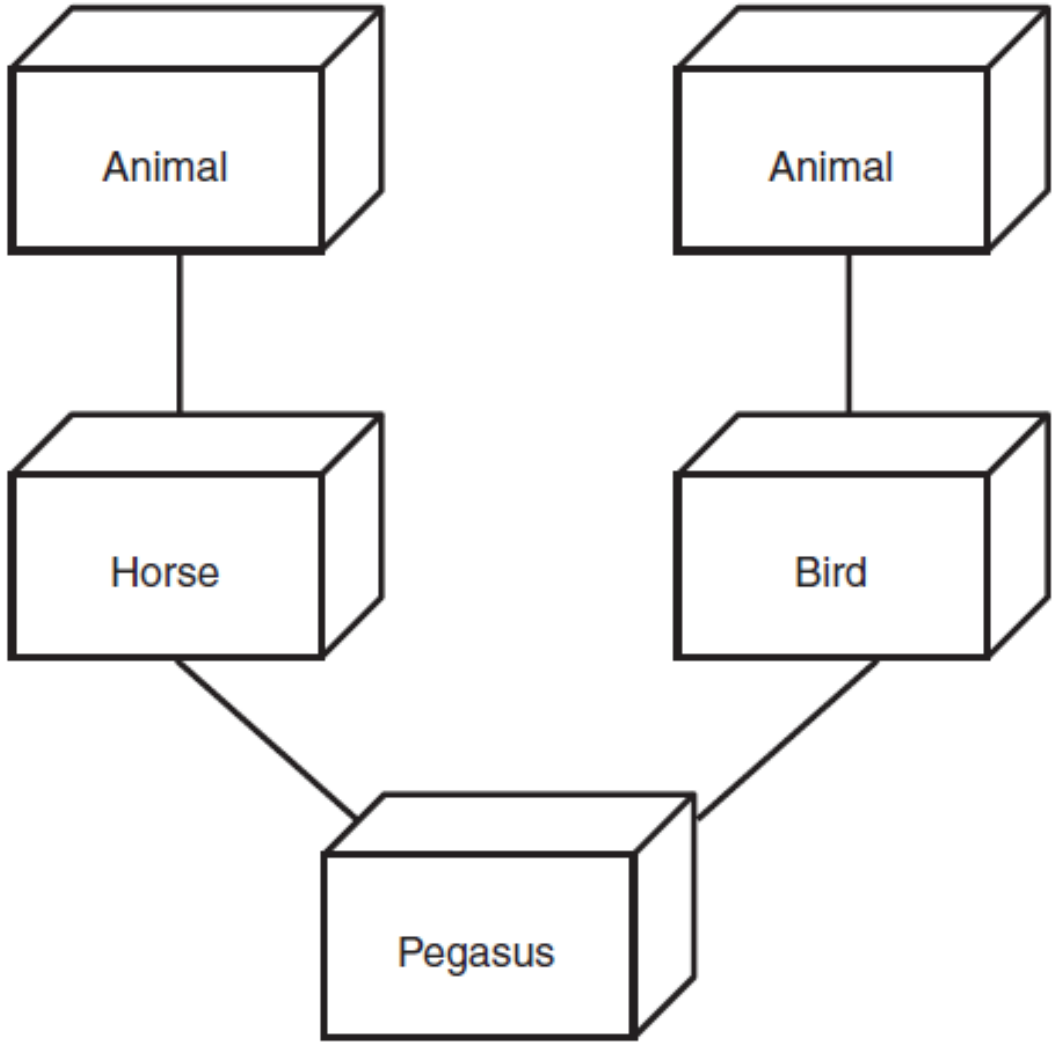
Horse constructor...
Bird constructor...
Pegasus constructor...
I can fly! I can fly! I can fly! Whinny!...
Your Pegasus is 5 hands tall and it does migrate.
A total of 10 people believe it exists.
Pegasus destructor...
Bird destructor...
Horse destructor...

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



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

Κοινή κλάση βάσης/2

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

Κοινή κλάση βάσης/3

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

Κοινή κλάση βάσης/4

```
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;   };
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/5

```
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...