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

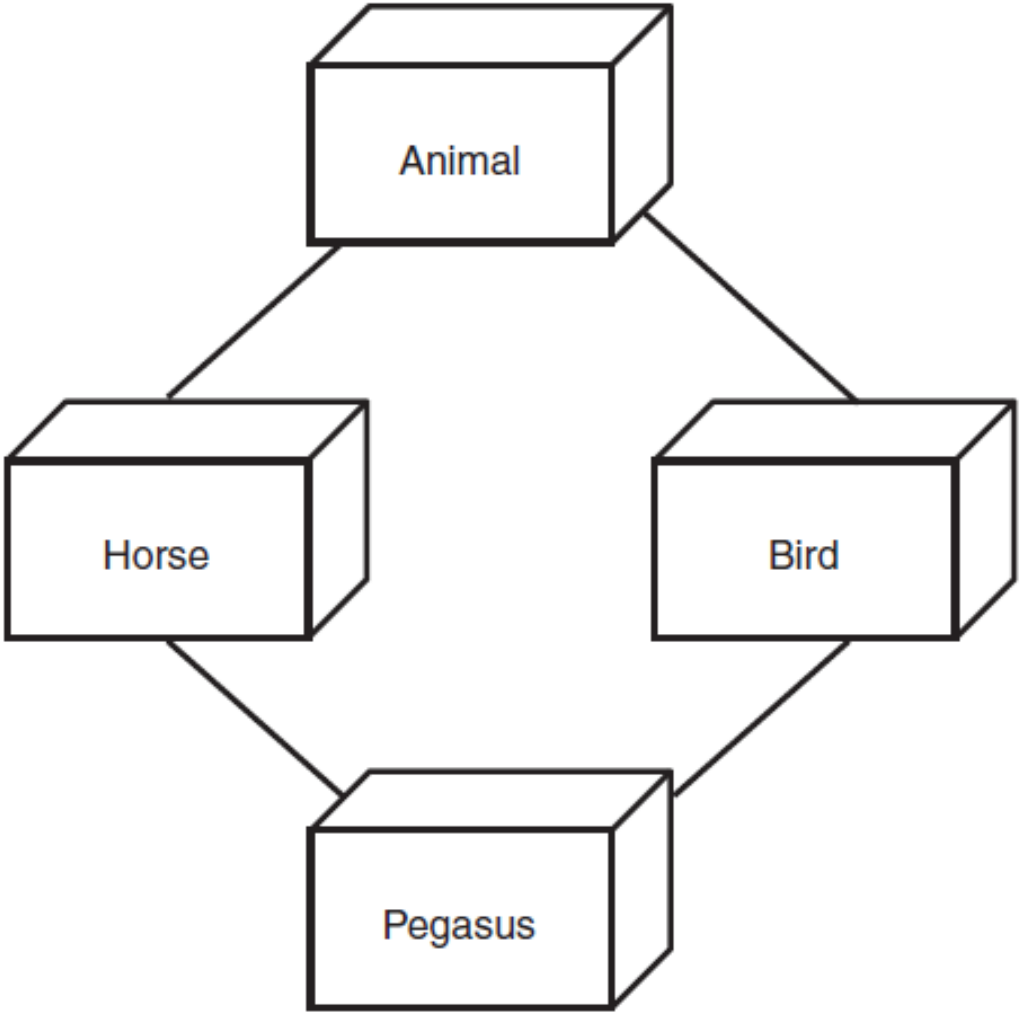
Σήμερα!

- ✓ Virtual Κληρονομικότητα
- ✓ Mixin classes
- ✓ Αφηρημένοι τύποι δεδομένων
- ✓ Pure Virtual συναρτήσεις

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



Virtual Κληρονομικότητα/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; }
protected:
  int itsAge; };

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

Virtual Κληρονομικότητα/2

```
class Horse : virtual 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";
}
```

Virtual Κληρονομικότητα/3

```
class Bird : virtual 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";
}
```

Virtual Κληρονομικότητα/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; }
private:
   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";    }
```

Virtual Κληρονομικότητα/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;
}
```

```
Animal constructor...
Horse constructor...
Bird constructor...
Pegasus constructor...
This pegasus is 4 years old.
Pegasus destructor...
Bird destructor...
Horse destructor...
Animal destructor...
```

Δήλωση κλάσεων για virtual κληρονομικότητα

- ✓ Για να διασφαλίσουμε ότι οι παραγόμενες κλάσεις διατηρούν ένα μόνο αντίγραφο κοινών συναρτήσεων των κλάσεων βάσης, δηλώνουμε τις κλάσεις βάσης ως παραγόμενες από μια virtual βάση.

Πολλαπλή Κληρονομικότητα

- ✓ Χρησιμοποιήστε Πολλαπλή Κληρονομικότητα όταν μία κλάση χρειάζεται συναρτήσεις και στοιχεία από περισσότερες της μιας κλάσης.
- ✓ Υπάρχουν compilers που δεν υποστηρίζουν την πολλαπλή κληρονομικότητα.
- ✓ Μη χρησιμοποιείται πολλαπλή κληρονομικότητα όταν δε χρειάζεται ή όταν σας καλύπτει η απλή κληρονομικότητα γιατί αυξάνει την πολυπλοκότητα του προγράμματος
- ✓ Αρχικοποιήστε τη virtual κλάση βάσης από την τελευταία στην ιεραρχία παραγόμενη κλάση.

Mixins classes

- ✓ Η Mixin class προσθέτει λειτουργικότητα ενώ δεν προσθέτει πολλά δεδομένα.
- ✓ Οι mixin classes προστίθενται σε μία παραγόμενη κλάση με public κληρονομικότητα.
- ✓ Αποτελούν συμβιβασμό μεταξύ απλής και πολλαπλής κληρονομικότητας.
- ✓ Η διαφορά τους από τις άλλες κλάσεις είναι ότι περιέχουν ελάχιστα ή καθόλου δεδομένα.

Αφηρημένοι τύποι δεδομένων/1

```
#include <iostream.h>
enum BOOL { FALSE, TRUE };

class Shape
  {
  public:
    Shape(){}
    ~Shape(){}
    virtual long GetArea() { return -1; }
    virtual long GetPerim() { return -1; }
    virtual void Draw() {}
 private:
 };
```

Αφηρημένοι τύποι δεδομένων/2

```
class Circle : public Shape
{
public:
     Circle(int radius):itsRadius(radius){}
     ~Circle(){}
     long GetArea() { return 3 * itsRadius * itsRadius; }
     long GetPerim() { return 9 * itsRadius; }
     void Draw();
private:
   int itsRadius;
   int itsCircumference;
};
void Circle::Draw()
{
   cout << "Circle drawing routine here!\n";
}
```

Αφηρημένοι τύποι δεδομένων/3

```
class Rectangle : public Shape    {
public:
      Rectangle(int len, int width):
        itsLength(len), itsWidth(width){}
      ~Rectangle(){}
      virtual long GetArea() { return itsLength * itsWidth; }
      virtual long GetPerim() {return 2*itsLength + 2*itsWidth; }
      virtual int GetLength() { return itsLength; }
      virtual int GetWidth() { return itsWidth; }
      virtual void Draw();
private:
   int itsWidth;      int itsLength;    };
void Rectangle::Draw()  {
   for (int i = 0; i<itsLength; i++)    {
      for (int j = 0; j<itsWidth; j++)   cout << "x ";
      cout << "\n";     }
}
```

Αφηρημένοι τύποι δεδομένων/4

```
class Square : public Rectangle    {
public:
      Square(int len);
      Square(int len, int width);
      ~Square(){}
      long GetPerim() {return 4 * GetLength();}
};
Square::Square(int len):   Rectangle(len,len)
{}
Square::Square(int len, int width):   Rectangle(len,width)
{
   if (GetLength() != GetWidth())
      cout << "Error, not a square... a Rectangle??\n";
}
```

Αφηρημένοι τύποι δεδομένων/5

```
int main()    {
   int choice;
   BOOL fQuit = FALSE;
   Shape * sp;
   while (1)     {
      cout << "(1)Circle (2)Rectangle (3)Square (0)Quit: ";
      cin >> choice;
      switch (choice)      {
         case 1: sp = new Circle(5);         break;
         case 2: sp = new Rectangle(4,6);    break;
         case 3: sp = new Square(5);         break;
         default: fQuit = TRUE;         break;         }
      if (fQuit)
         break;
      sp->Draw();
      cout << "\n";      }
  return 0; }
```

Αφηρημένοι τύποι δεδομένων/ έξοδος

(1)Circle (2)Rectangle (3)Square (0)Quit: 3
x x x x x x
x x x x x x
x x x x x x
x x x x x x

(1)Circle (2)Rectangle (3)Square (0)Quit:2
x x x x x
x x x x x
x x x x x
x x x x x
x x x x x

(1)Circle (2)Rectangle (3)Square (0)Quit:0

Pure Virtual συναρτήσεις

- ✓ Μια virtual συνάρτηση γίνεται pure virtual όταν αρχικοποιείται στο 0.

```
virtual void Draw() = 0;
```

- ✓ Κάθε κλάση με μία ή περισσότερες Pure Virtual συναρτήσεις αποτελεί Αφηρημένη κλάση
- ✓ Δεν μπορούμε να δημιουργήσουμε αντικείμενο αφηρημένης κλάσης
- ✓ Είναι υποχρεωτικό να ορίσουμε τις Pure Virtual συναρτήσεις σε κάθε παραγόμενη κλάση

Pure Virtual συναρτήσεις

```
class Shape
{
public:
    Shape(){}
    ~Shape(){}
    virtual long GetArea() = 0;
    virtual long GetPerim()= 0;
    virtual void Draw() = 0;
private:
};
```

```
(1)Circle (2)Rectangle (3)Square (0)Quit: 2
x x x x x x
x x x x x x
x x x x x x
x x x x x x

(1)Circle (2)Rectangle (3)Square (0)Quit: 3
x x x x x
x x x x x
x x x x x
x x x x x
x x x x x

(1)Circle (2)Rectangle (3)Square (0)Quit: 0
```

Pure Virtual συναρτήσεις/1

```
#include <iostream.h>
enum BOOL { FALSE, TRUE };
class Shape
{
public:
  Shape(){}
  ~Shape(){}
  virtual long GetArea() = 0;
  virtual long GetPerim()= 0;
  virtual void Draw() = 0;
private:
};
 void Shape::Draw()
{
  cout << "Abstract drawing mechanism!\n";
}
```

Pure Virtual συναρτήσεις/2

```
class Circle : public Shape    {
public:
     Circle(int radius):itsRadius(radius){}
     ~Circle(){}
     long GetArea() { return 3 * itsRadius * itsRadius; }
     long GetPerim() { return 9 * itsRadius; }
     void Draw();
private:
   int itsRadius;
   int itsCircumference;
};
void Circle::Draw()
{
   cout << "Circle drawing routine here!\n";
   Shape::Draw();
}
```

Pure Virtual συναρτήσεις/3

```
class Rectangle : public Shape    {
public:
      Rectangle(int len, int width):
        itsLength(len), itsWidth(width){}
      ~Rectangle(){}
      long GetArea() { return itsLength * itsWidth; }
      long GetPerim() {return 2*itsLength + 2*itsWidth; }
      virtual int GetLength() { return itsLength; }
      virtual int GetWidth() { return itsWidth; }
      void Draw();
private:
   int itsWidth;    int itsLength;    };
void Rectangle::Draw()    {
   for (int i = 0; i<itsLength; i++)     {
      for (int j = 0; j<itsWidth; j++)     cout << "x ";
      cout << "\n";     }
    Shape::Draw();   }
```

Pure Virtual συναρτήσεις/4

```
class Square : public Rectangle   {
public:
      Square(int len);
      Square(int len, int width);
      ~Square(){}
      long GetPerim() {return 4 * GetLength();}
};
Square::Square(int len):      Rectangle(len,len)
{}
Square::Square(int len, int width):   Rectangle(len,width)
{
   if (GetLength() != GetWidth())
      cout << "Error, not a square... a Rectangle??\n";
}
```

Pure Virtual συναρτήσεις/5

```
int main()    {
   int choice;
   BOOL fQuit = FALSE;
   Shape * sp;
   while (1)      {
     cout << "(1)Circle (2)Rectangle (3)Square (0)Quit: ";
     cin >> choice;
     switch (choice)      {
        case 1: sp = new Circle(5);        break;
        case 2: sp = new Rectangle(4,6);        break;
        case 3: sp = new Square (5);       break;
        default: fQuit = TRUE;      break;    }
     if (fQuit)     break;
     sp->Draw();
     cout << "\n";   }
  return 0;
}
```

Pure Virtual συναρτήσεις/έξοδος

(1)Circle (2)Rectangle (3)Square (0)Quit: 2
x x x x x x
x x x x x x
x x x x x x
x x x x x x
Abstract drawing mechanism!

(1)Circle (2)Rectangle (3)Square (0)Quit: 3
x x x x x
x x x x x
x x x x x
x x x x x
x x x x x
Abstract drawing mechanism!

(1)Circle (2)Rectangle (3)Square (0)Quit: 0