Πανεπιστήμιο Αιγαίου

Μεθοδολογίες και Γλώσσες Προγραμματισμού Ι

Κληρονομικότητα

Εργίνα Καβαλλιεράτου (kavallieratou@aegean.gr)

Μόνιμη Επίκουρος Καθηγήτρια

Τμήμα Μηχανικών Πληροφοριακών & Επικοινωνιακών Συστημάτων

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

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!

- ✓ Overriding
- ✓ Overloading Vs Overriding
- ✓ Απόκρυψη συναρτήσεων
- ✓ Κλήση overridden συνάρτησης
- ✓ Virtual Συναρτήσεις
- ✓ Abstract Classes
- ✓ Κανόνες πρόσβασης Κληρονομικότητας

Υπερίσχυση Συναρτήσεων (overriding)

- ✓ Έχουμε υπερίσχυση συναρτήσεων όταν στην παράγωγη κλάση ξαναδημιουργείται μία συνάρτηση της βάσης με τον ίδιο τύπο επιστροφής, όνομα και παραμέτρους και διαφορετική υλοποίηση.

- ✓ Όταν καλείται ένα αντικείμενο της παραγόμενης κλάσης καλείται η νέα συνάρτηση.

Overriding/1

```
#include <iostream.h>
enum BREED{YORKIE,CAIRN,DANDIE,SHETLAND,DOBERMAN,LAB};
class Mammal  {
public:
  Mammal() { cout << "Mammal constructor...\n"; }
 ~Mammal() { cout << "Mammal destructor...\n"; }
  int GetAge()const { return itsAge; }
  void SetAge(int age) { itsAge = age; }
  int GetWeight() const { return itsWeight; }
  void SetWeight(int weight) { itsWeight = weight; }
  void Speak()const { cout << "Mammal sound!\n"; }
  void Sleep()const { cout << "shhh. I'm sleeping.\n"; }
protected:
  int itsAge;
  int itsWeight;  };
```

Overriding/2

```
class Dog : public Mammal
{
public:
   Dog(){ cout << "Dog constructor...\n"; }
   ~Dog(){ cout << "Dog destructor...\n"; }
   BREED GetBreed() const { return itsBreed; }
   void SetBreed(BREED breed) { itsBreed = breed; }
   void WagTail() { cout << "Tail wagging...\n"; }
   void BegForFood() { cout << "Begging for food...\n"; }
   void Speak()const { cout << "Woof!\n"; }
private:
   BREED itsBreed;
};
```

Overriding/3

```
int main()
{
   Mammal bigAnimal;
   Dog fido;
   bigAnimal.Speak();
   fido.Speak();
 return 0;
}
```

```
Mammal constructor...
Mammal constructor...
Dog constructor...
Mammal sound!
Woof!
Dog de
structor...
Mammal destructor...
Mammal destructor...
```

Overloading Vs Overriding

- ✓ Αυτοί οι όροι μοιάζουν και κάνουν παρόμοια πράγματα
- ✓ Όταν κάνουμε μία συνάρτηση overloading, δημιουργούμε μία συνάρτηση με το ίδιο όνομα αλλά διαφορετικές παραμέτρους.
- ✓ Το overriding δημιουργεί στην παραγόμενη κλάση μία συνάρτηση με τα ίδια όνομα, παραμέτρους και τύπο επιστροφής.

Απόκρυψη συναρτήσεων/1

```
#include <iostream.h>
class Mammal
{
 public:
    void Move() const { cout << "Mammal move one step\n"; }
    void Move(int distance) const
   {
      cout << "Mammal move ";
      cout << distance <<" _steps.\n";
   }
 protected:
    int itsAge;
    int itsWeight;
};
```

Απόκρυψη συναρτήσεων/2

```
class Dog : public Mammal   {
public:
   void Move() const { cout << "Dog move 5 steps.\n"; }
};
int main()
{
   Mammal bigAnimal;
   Dog fido;
   bigAnimal.Move();
   bigAnimal.Move(2);
   fido.Move();
   // fido.Move(10);
  return 0;
}
```

```
Mammal move one step
Mammal move 2 steps.
Dog move 5 steps.
```

Απόκρυψη συναρτήσεων

- ✓ Συμβαίνει απόκρυψη συνάρτησης όταν παραλείψουμε οποιοδήποτε τμήμα της κεφαλίδας της, ακόμα και αν είναι μόνο η λέξη const

- ✓ Αν έχουμε overriding σε μία συνάρτηση της βάσης, μπορούμε και πάλι να την καλέσουμε αν γράψουμε το πλήρες όνομα:

Mammal::Move()

Κλήση overridden συνάρτησης/1

```
#include <iostream.h>
class Mammal
  {
   public:
     void Move() const { cout << "Mammal move one step\n"; }
     void Move(int distance) const
    {
      cout << "Mammal move ";
      cout << distance <<" _steps.\n";
    }
   protected:
     int itsAge;
     int itsWeight;
  };
```

Κλήση overridden συνάρτησης/2

```
class Dog : public Mammal   {
public:
   void Move()const;
};
void Dog::Move() const {
   cout << "In dog move...\n";
   Mammal::Move(3);   }
int main()   {
   Mammal bigAnimal;
    Dog fido;
   bigAnimal.Move(2);
   fido.Mammal::Move(6);
 return 0; }
```

```
Mammal move 2 steps.
Mammal move 6 steps.
```

Virtual Συναρτήσεις

- ✓ Μία Virtual συνάρτηση είναι συνάρτηση της βασικής κλάσης που μπορεί να υπερκαλυφθεί από συνάρτηση της παραγόμενης κλάσης
- ✓ Σύνταξη

  virtual ret_type FunctionName(args)

Virtual συναρτήσεις/1

```
#include <iostream.h>
class Mammal
{
 public:
   Mammal():itsAge(1) { cout << "Mammal constructor...\n"; }
   ~Mammal() { cout << "Mammal destructor...\n"; }
   void Move() const { cout << "Mammal move one step\n"; }
   virtual void Speak() const { cout << "Mammal speak!\n"; }
protected:
   int itsAge;
};
```

Virtual συναρτήσεις/2

```
class Dog : public Mammal   {
public:
   Dog() { cout << "Dog Constructor...\n"; }
   ~Dog() { cout << "Dog destructor...\n"; }
   void WagTail() { cout << "Wagging Tail...\n"; }
   void Speak()const { cout << "Woof!\n"; }
   void Move()const { cout << "Dog moves 5 steps...\n"; }  };
int main()    {
   Mammal *pDog = new Dog;
   pDog->Move();
   pDog->Speak();
  return 0;
 }
```

```
Mammal constructor...
Dog Constructor...
Mammal move one step
Woof!
```

Virtual συναρτήσεις

- ✓ Στις μη-εικονικές συναρτήσεις η διεύθυνση του κώδικα της συνάρτησης που θα κληθεί είναι συγκεκριμένη και γνωστή κατά το στάδιο της μεταγλώττισης (**στατική σύνδεση - static binding**).
- ✓ Στις εικονικές (virtual) συναρτήσεις προσδιορίζεται κατά την ώρα εκτέλεσης του προγράμματος (**δυναμική σύνδεση – dynamic binding**)

Abstract Classes

- ✓ Μια κλάση που έχει τουλάχιστο μία καθαρά virtual συνάρτηση (**pure virtual function**) λέγεται αφαιρετική κλάση (abstract class).

```
class Mammal  // abstract class
  {
   public:
     …
     virtual void Speak() =0;        //pure virtual function
     …
   };
```

Πολλαπλές Virtual συναρτήσεις/1

```
#include <iostream.h>
class Mammal    {
public:
   Mammal():itsAge(1) {  }
   ~Mammal() { }
   virtual void Speak() const { cout << "Mammal speak!\n"; }
protected:
   int itsAge;
};
```

Πολλαπλές Virtual συναρτήσεις/2

```
class Dog : public Mammal   {
  public:
    void Speak()const { cout << "Woof!\n"; }    };
class Cat : public Mammal    {
  public:
    void Speak()const { cout << "Meow!\n"; }   };
 class Horse : public Mammal  {
  public:
    void Speak()const { cout << "Winnie!\n"; } };
 class Pig : public Mammal    {
  public:
    void Speak()const { cout << "Oink!\n"; }   };
```

Πολλαπλές Virtual συναρτήσεις/3

```
int main()   {
   Mammal* theArray[5];
   Mammal* ptr;
   int choice, i;
   for ( i = 0; i<5; i++)    {
      cout << "(1)dog (2)cat (3)horse (4)pig: ";
      cin >> choice;
      switch (choice)    {
         case 1: ptr = new Dog;    break;
         case 2: ptr = new Cat;     break;
         case 3: ptr = new Horse;    break;
         case 4: ptr = new Pig;     break;
         default: ptr = new Mammal;    break;    }
      theArray[i] = ptr;   }
   for (i=0;i<5;i++)    theArray[i]->Speak();
return 0; }
```

Κανόνες πρόσβασης Κληρονομικότητας

- ✓ Τα **private** μέλη δεν είναι ορατά εκτός κλάσης
- ✓ Τα **protected** μέλη κληρονομούνται, αλλά δεν είναι ορατά εκτός κλάσης
- ✓ Η C++ έχει 3 επίπεδα ελέγχου πρόσβασης
- ✓ Σύνταξη: **class B *: είδος πρόσβασης* A {...};**
- ✓ Τα τρία επίπεδα είναι:

  public: **public** παραμένει **public**, **protected** παραμένει **protected**

  protected: **public** γίνεται **protected**, **protected** μένει **protected**

  private: **public** and **protected** γίνονται **private**

Κανόνες πρόσβασης Κληρονομικότητας

Κληρονομικότητα με **ιδιωτική-private** πρόσβαση

Μέλη κλάσης βάσης	Πρόσβαση από αντικείμενα της κλάσης βάσης	Πρόσβαση από την παράγωγη κλάση	Πρόσβαση από αντικείμενα της παράγωγης κλάσης
public	ΝΑΙ	ΝΑΙ	ΟΧΙ
private	ΟΧΙ	ΟΧΙ	ΟΧΙ
protected	ΟΧΙ	ΝΑΙ	ΟΧΙ

Ο παραπάνω πίνακας προσβάσεων είναι ίδιος και για την περίπτωση της προστατευμένης (protected) πρόσβασης. Διαφορά μπορούμε να αντιληφθούμε στην περίπτωση **πολλαπλής κληρονομικότητας**, όπου η παράγωγη κλάση αποτελεί την κλάση βάσης για την παραγωγή μιας νέας κλάσης.

Δηλώσεις Πρόσβασης

- ✓ Στην κληρονομικότητα μπορεί να ξαναδηλωθεί η πρόσβαση
- ✓ Η νέα πρόσβαση δεν μπορεί να είναι πιο μεγάλη

```
class A {
 protected: int vprot;
 public: int prot;
};

class B : public A {
 protected:
  A::prot; // δήλωση πρόσβασης;
 };
```

Παράδειγμα δείκτες & αναφορές/1

```
#include <iostream.h>
enum BOOL { FALSE, TRUE };
class Car {
public:
 Car():itsAge(1) {  }
   ~ Car() { }
   virtual void Move() const { cout << “Car moves!\n"; }
protected:
   int itsAge;    };

class Truck : public Car   {
public:
   void Move()const { cout << “Truck moves!\n"; }
};
```

Παράδειγμα δείκτες & αναφορές/2

```
class Bus: public Car   {
public:
   void Moves()const { cout << “Bus moves!\n"; }    };
void ValueFunction (Car);
void PtrFunction   (Car *);
void RefFunction (Car &);
int main()   {
  Car * ptr=0;
   int choice;
   while (1)
   {
      BOOL fQuit = FALSE;
      cout << "(1)Truck (2)Bus (0)Quit: ";
      cin >> choice;
```

Παράδειγμα δείκτες & αναφορές/3

```
switch (choice)      {
    case 0: fQuit = TRUE;
    break;
    case 1: ptr = new Truck;
    break;
    case 2: ptr = new Bus;
    break;
    default: ptr = new Car;
    break;          }
  if (fQuit)       break;
   PtrFunction(ptr);
   RefFunction(*ptr);
   ValueFunction(*ptr);      }
return 0;   }
```

Παράδειγμα δείκτες & αναφορές/4

```
void ValueFunction (Car CarValue)
{
  CarValue.Move();
}

void PtrFunction (Car * pCar)
{
  pCar->Move();
}

void RefFunction (Car & rCar)
{
  rCar.Move();
}
```

Παράδειγμα δείκτες & αναφορές - έξοδος

(1)Truck (2)Bus (0)Quit: 1

Truck moves!

Truck moves!

Car moves!

(1)Truck (2)Bus (0)Quit: 2

Bus moves!

Bus moves!

Car moves!

(1)Truck (2)Bus (0)Quit: 0