Κληρονομικότητα

Εργίνα Καβαλλιεράτου

Σημερινό Μάθημα

- ✓ Κλάση Βάσης/Παράγωγη κλάση
- ✓ Απλή κληρονομικότητα
- ✓ Protected δεδομένα
- ✓ Overloading
- ✓ Overriding
- ✓ Απόκρυψη συναρτήσεων
- ✓ Κλήση overridden συνάρτησης
- ✓ Virtual Συναρτήσεις
- ✓ Abstract Classes
- ✓ Κανόνες πρόσβασης Κληρονομικότητας
- ✓ Copy Constructor

Κλάση Βάση/Παράγωγη

- ✓ Τα διάφορα αντικείμενα μπορούν να έχουν μεταξύ τους σχέση ταξινόμησης π.χ Θηλαστικό - Σκύλος
- ✓ Η C++ δίνει τη δυνατότητα απεικόνισης τέτοιων σχέσεων, ορίζοντας κλάσεις παραγόμενες από άλλες.
- ✓ Για παράδειγμα μπορούμε να ορίσουμε την κλάση Σκύλος ως παράγωγη της κλάσης Θηλαστικό.
- ✓ Σε αυτή την περίπτωση π.χ δεν χρειάζεται να ορίσουμε ότι ο Σκύλος κινείται αν έχουμε ήδη ορίσει ότι το Θηλαστικό κινείται.
- ✓ Η κλάση Θηλαστικό λέγεται Βάση και η Σκύλος Παράγωγη.
- ✓ Μία κλάση Βάση μπορεί να έχει πολλές Παράγωγες.

Παραγόμενες κλάσεις

- ✓ Ας φανταστούμε ότι πρέπει να φτιάξουμε ένα πρόγραμμα που περιγράφει μία φάρμα ζώων.
- ✓ Θα πρέπει να συμπεριλάβουμε κλάσεις για τα διάφορα ζώα.
- ✓ Όταν δηλώνουμε μία κλάση θα πρέπει να υποδεικνύουμε από ποια κλάση παράγεται, γράφοντας μετά το όνομα της κλάσης, τον τρόπο παραγωγής:

```
class Dog : public Mammal
```

- ✓ Η κλάση Βάση **ΠΡΕΠΕΙ** να έχει δηλωθεί νωρίτερα.

Απλή κληρονομικότητα/1

```
#include <iostream.h>
  enum BREED { YORKIE, CAIRN,DANDIE,SHETLAND,DOBERMAN,LAB };
  class Mammal    {
  public:
    Mammal();
    ~Mammal();
    int GetAge()const;        void SetAge(int);
    int GetWeight() const;        void SetWeight();
    void Speak();
    void Sleep();
  protected:
    int itsAge;
    int itsWeight;  };
```

Απλή κληρονομικότητα/2

```
class Dog : public Mammal
  {
  public:
     Dog();
     ~Dog();
     BREED GetBreed() const;
     void SetBreed(BREED);
     WagTail();
     BegForFood();
  protected:
     BREED itsBreed;
};
```

Protected δεδομένα

- ✓ Η παραγόμενη κλάση, κληρονομεί από την κλάση βάσης όλα τα δεδομένα και τις συναρτήσεις, **εκτός** από τον τελεστή αντιγραφής, το δομητή και τον αποδομητή.
- ✓ Τα private μέλη δεν είναι διαθέσιμα στις παραγόμενες κλάσεις
- ✓ Τα protected μέλη είναι απολύτως διαθέσιμα στις παραγόμενες κλάσεις και private για το υπόλοιπο πρόγραμμα.
- ✓ Οι συναρτήσεις μέλη έχουν πρόσβαση σε όλα τα δεδομένα και συναρτήσεις της κλάσης τους (ακόμα και τα private) και στα public και protected μέλη των κλάσεων βάσης.

Παραγόμενο αντικείμενο/1

```
#include <iostream.h>
   enum BREED{YORKIE,CAIRN,DANDIE,SHETLAND,DOBERMAN,LAB};
  class Mammal  {
   public:
     Mammal():itsAge(2), itsWeight(5){}
     ~Mammal(){}
     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 << "I'm sleeping.\n"; }
  protected:
     int itsAge;       int itsWeight;  };
```

Παραγόμενο αντικείμενο/2

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

Παραγόμενο αντικείμενο/3

```
int main()
  {
    Dog fido;
    fido.Speak();
    fido.WagTail();
    cout << "Fido is " << fido.GetAge() << " years old\n";
   return 0;
  }
```

```
Mammal sound!
Tail wagging...
Fido is 2 years old
```

Constructors & Destructors

- ✓Ένα αντικείμενο Dog είναι και αντικείμενο Mammal.
- ✓Όταν δημιουργείται το Fido, πρώτα καλείται ο δομητής της βάσης και μετά του Dog.
- ✓Όταν το Fido καταστρέφεται πρώτα καλείται ο αποδομητής του Dog και μετά του Mammal.

Constructors & Destructors/1

```
#include <iostream.h>
 enum BREED{YORKIE,CAIRN,DANDIE,SHETLAND,DOBERMAN,LAB};
class Mammal  {
 public:
   Mammal();      ~Mammal();
   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;  };
```

Constructors & Destructors/2

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

Constructors & Destructors/3

```
Mammal::Mammal(): itsAge(1), itsWeight(5)  {cout << "Mammal constructor...\n";  }
Mammal::~Mammal()   { cout << "Mammal destructor...\n";  }
Dog::Dog():  itsBreed(YORKIE)   { cout << "Dog constructor...\n";  }
Dog::~Dog()  { cout << "Dog destructor...\n";   }
int main()    {
  Dog fido;
  fido.Speak();
  fido.WagTail();
  cout << "Fido is " << fido.GetAge() << " years old\n";
 return 0; }
```

```
Mammal constructor...
Dog constructor...
Mammal sound!
Tail wagging...
Fido is 1 years old
Dog destructor...
Mammal destructor...
```

Constructor overloading/1

```
#include <iostream.h>
 enum BREED{YORKIE,CAIRN,DANDIE,SHETLAND,DOBERMAN,LAB};
class Mammal  {
 public:
  Mammal();
  Mammal::Mammal(int age):  itsAge(age),  itsWeight(5);
  ~Mammal();
  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 << "sleeping.\n"; }
protected:
```

Constructor overloading/2

```
class Dog : public Mammal    {
  public:
    Dog();       Dog(int age);
    Dog(int age, int weight);       Dog(int age, BREED breed);
    Dog(int age, int weight, BREED breed);       ~Dog();
    BREED GetBreed() const { return itsBreed; }
    void SetBreed(BREED breed) { itsBreed = breed; }
    void WagTail() { cout << "Tail wagging...\n"; }
    void BegForFood() { cout << "Begging for food...\n"; }
  private:
    BREED itsBreed;  };
```

Constructor overloading/3

```
Mammal::Mammal():   itsAge(1),  itsWeight(5)   {  cout << "Mammal constructor...\n";  }
Mammal::Mammal(int age):  itsAge(age),  itsWeight(5)  {
   cout << "Mammal(int) constructor...\n";  }
Mammal::~Mammal()  {cout << "Mammal destructor...\n";  }
 Dog::Dog(): Mammal(), itsBreed(YORKIE)  {
   cout << "Dog constructor...\n";  }
 Dog::Dog(int age): Mammal(age), itsBreed(YORKIE) {
   cout << "Dog(int) constructor...\n"; }
 Dog::Dog(int age, int weight):  Mammal(age), itsBreed(YORKIE) {
   itsWeight = weight;
   cout << "Dog(int, int) constructor...\n";  }
```

Constructor overloading/4

```
Dog::Dog(int age, int weight, BREED breed): Mammal(age), itsBreed(breed){
      itsWeight = weight;
      cout << "Dog(int, int, BREED) constructor...\n";  }
Dog::Dog(int age, BREED breed):  Mammal(age), itsBreed(breed)  {
     cout << "Dog(int, BREED) constructor...\n";   }
Dog::~Dog()  {
     cout << "Dog destructor...\n";  }
```

Constructor overloading/5

```
int main()
{
   Dog fido;     Dog rover(5);
   Dog buster(6,8);
   Dog yorkie (3,YORKIE);
   Dog dobbie (4,20,DOBERMAN);
   fido.Speak();
   rover.WagTail();
   cout << "Yorkie is " << yorkie.GetAge() << " years old\n";
    cout << "Dobbie weighs ";
    cout << dobbie.GetWeight() << "pounds\n";
   return 0; }
```

```
Mammal constructor...
Dog constructor...
Mammal(int) constructor...
Dog(int) constructor...
Mammal(int) constructor...
Dog(int, int) constructor...
Mammal(int) constructor...
Dog(int, BREED) constructor....
Mammal(int) constructor...
Dog(int, int, BREED) constructor...
Mammal sound!
Tail wagging...
Yorkie is 3 years old.
Dobbie weighs 20 pounds.
Dog destructor. . .
Mammal destructor...
Dog destructor...
Mammal destructor...
Dog destructor...
Mammal destructor...
Dog destructor...
Mammal destructor...
Dog destructor...
Mammal destructor...
```

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 << "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 destructor...
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!
```

Abstract Classes

- ✓Μια κλάση που έχει τουλάχιστο μία καθαρά virtual συνάρτηση λέγεται αφαίρετική κλάση (abstract class).

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    {
    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; }
```

```
(1)dog (2)cat (3)horse (4)pig: 1
(1)dog (2)cat (3)horse (4)pig: 2
(1)dog (2)cat (3)horse (4)pig: 3
(1)dog (2)cat (3)horse (4)pig: 4
(1)dog (2)cat (3)horse (4)pig: 5
Woof!
Meow!
Winnie!
Oink!
Mammal speak!
```

Κανόνες πρόσβασης Κληρονομικότητας

- ✓ Τα **private** μέλη δεν κληρονομούνται
- ✓ 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**

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;
 };
```

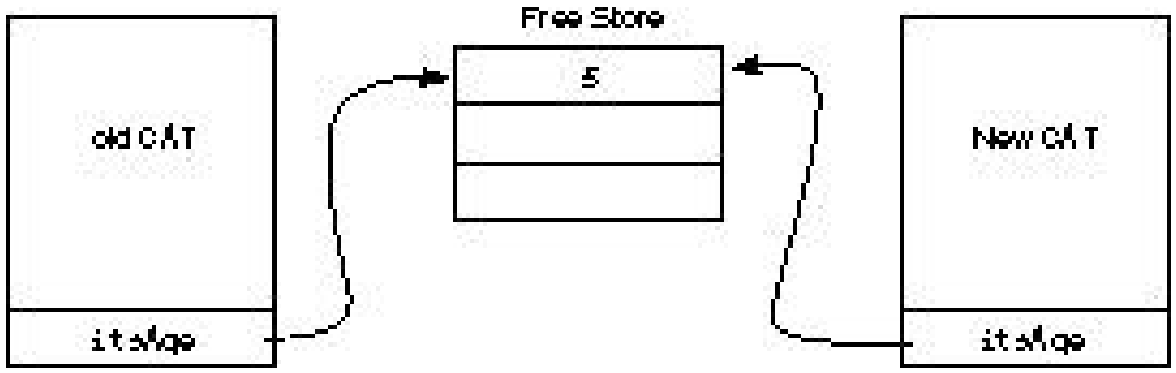
Copy Constructor

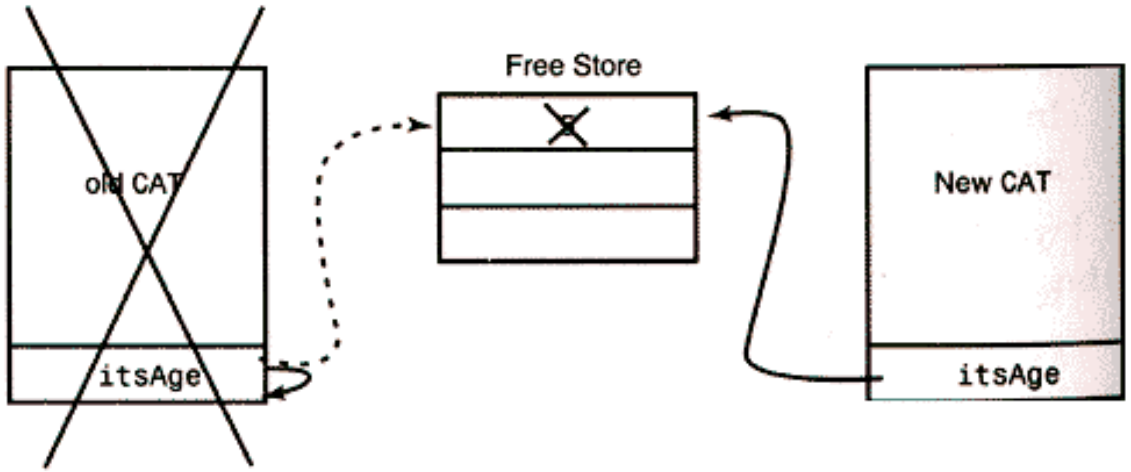
- ✓ Ο compiler διαθέτει τον Copy Constructor για τις περιπτώσεις που χρειαζόμαστε αντίγραφο αντικειμένου, πχ πέρασμα σε συνάρτηση
- ✓ Όλοι οι copy constructors παίρνουν ως παράμετρο μια αναφορά σε αντικείμενο της ίδιας κλάσης.

CAT(const CAT & theCat);

- ✓ Ο default copy constructor αντιγράφει κάθε δεδομένο του αντικειμένου-παραμέτρου στα δεδομένα ενός νέου αντικειμένου.
- ✓ Αυτό μπορεί να είναι πρόβλημα όταν τα δεδομένα είναι δείκτες καθώς θα δείχνουν και οι δύο στην ίδια θέση μνήμης αλλά θα πάψουν να υπάρχουν με την καταστροφή του αντίστοιχου αντικειμένου

Copy Constructor





Copy Constructor

- ✓Η λύση είναι να δημιουργήσουμε το δικό μας copy constructor που θα δεσμεύσει την απαιτούμενη μνήμη εξ’ αρχής πριν αντιγράψει τις τιμές σε νέα μνήμη.

Copy Constructor

```
#include <iostream.h>
class CAT {
public:
   CAT();
   CAT (const CAT &);
   ~CAT();
   int GetAge() const { return *itsAge; }
   int GetWeight() const { return *itsWeight; }
   void SetAge(int age) { *itsAge = age; }
private:
   int *itsAge;       int *itsWeight; };
```

Copy Constructor/2

```
CAT::CAT()  {
   itsAge = new int;
   itsWeight = new int;
   *itsAge = 5;       *itsWeight = 9;  }
CAT::CAT(const CAT & rhs)  {
   itsAge = new int;     itsWeight = new int;
   *itsAge = rhs.GetAge();
   *itsWeight = rhs.GetWeight();  }
CAT::~CAT()  {
   delete itsAge;   itsAge = 0;
   delete itsWeight;  itsWeight = 0;  }
```

Copy Constructor/3

```
int main()  {
    CAT frisky;
    cout << "frisky's age: " << frisky.GetAge() << endl;
    cout << "Setting frisky to 6...\n";    frisky.SetAge(6);
    cout << "Creating boots from frisky\n"; CAT boots(frisky);
    cout << "frisky's age: " << frisky.GetAge() << endl;
    cout << "boots' age: " << boots.GetAge() << endl;
    cout << "setting frisky to 7...\n";    frisky.SetAge(7);
    cout << "frisky's age: " << frisky.GetAge() << endl;
    cout << "boot's age: " << boots.GetAge() << endl;
    return 0;  }
```

```
frisky's age: 5
Setting frisky to 6...
Creating boots from frisky
frisky's age: 6
boots' age: 6
setting frisky to 7...
frisky's age: 7
boots' age: 6
```