ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΤΕΧΝΙΚΕΣ 2025-2026

https://helios.ntua.gr/course/view.php?id=869

6/3/2026

Διδάσκοντες:

Νίκος Παπασπύρου, Καθ.
Βασίλης Βεσκούκης, Καθ.
Νίκος Λεονάρδος, Επ. Καθ.
Πέτρος Ποτίκας, ΕΔΙΠ
Γιώργος Σιόλας, ΕΔΙΠ
Σωτήρης Κοκόσης, ΕΔΙΠ

progtech@cslab.ece.ntua.gr

Διαφάνειες παρουσιάσεων

- Η γλώσσα προγραμματισμού C++
- Αρχές αντικειμενοστρεφούς προγραμματισμού
- Δομές δεδομένων

C++

TEMPLATES

C++ Templates

- Γενικός προγραμματισμός (generic programming)
- Function templates: πράξεις που εφαρμόζονται σε (σχεδόν) οποιονδήποτε τύπο δεδομένων
- Class templates: τύποι δεδομένων που περιέχουν ή εξαρτώνται από (σχεδόν) οποιονδήποτε τύπο δεδομένων

C++ Templates

- Γενικός προγραμματισμός (generic programming)
- Function templates: πράξεις που εφαρμόζονται σε (σχεδόν) οποιονδήποτε τύπο δεδομένων
- Class templates: τύποι δεδομένων που περιέχουν ή εξαρτώνται από (σχεδόν) οποιονδήποτε τύπο δεδομένων

- Παράδειγμα: αλγόριθμος ταξινόμησης που εφαρμόζεται σε δεδομένα διαφορετικών τύπων

Function templates (i)

- Ταξινόμηση ακεραίων με bubble sort

```
void bubble_sort(int n, int a[]) {
  for (int i = 0; i < n-1; ++i)
    for (int j = n-2; j >= i; --j)
      if (a[j] > a[j+1]) {
        int t = a[j];
        a[j] = a[j+1];
        a[j+1] = t;
      }
}
int a[] = { 42, 17, 4, 3, 8, 2, 1, 9 };
int na = sizeof(a) / sizeof(a[0]);
bubble_sort(na, a);
```

Function templates (ii)

- Ταξινόμηση συμβολοσειρών με bubble sort

```
void bubble_sort(int n, string a[]) {
  for (int i = 0; i < n-1; ++i)
    for (int j = n-2; j >= i; --j)
      if (a[j] > a[j+1]) {
        string t = a[j];
        a[j] = a[j+1];
        a[j+1] = t;
      }
}
string b[] = {"ba", "abc", "aa", "bab"};
int nb = sizeof(b) / sizeof(b[0]);
bubble_sort(nb, b);
```

Function templates (iii)

```
template <typename T>
void bubble_sort(int n, T a[]) {
  for (int i = 0; i < n-1; ++i)
    for (int j = n-2; j >= i; --j)
      if (a[j] > a[j+1]) {
        T t = a[j];
        a[j] = a[j+1];
        a[j+1] = t;
      }
}
```

- Για (σχεδόν) οποιονδήποτε τύπο **T**, φανταστείτε τον σαν παράμετρο

Function templates (iv)

```
template <typename T>
void bubble_sort(int n, T a[]) {
  for (int i = 0; i < n-1; ++i)
    for (int j = n-2; j >= i; --j)
      if (a[j] > a[j+1]) {
        T t = a[j];
        a[j] = a[j+1];
        a[j+1] = t;
      }
}
```

- Απαιτείται `operator>` (σύγκριση)
  και `operator=` (ανάθεση)

Function templates (v)

```
int a[] = { 42, 17, 4, 3, 8, 2, 1, 9 };
int na = sizeof(a) / sizeof(a[0]);
bubble_sort(na, a);                    // T = int


string b[] = {"ba", "abc", "aa", "bab"};
int nb = sizeof(b) / sizeof(b[0]);
bubble_sort(nb, b);                    // T = string
```

- Για (σχεδόν) οποιονδήποτε τύπο **T**, φανταστείτε τον σαν παράμετρο
- Γιατί «σχεδόν»;

Class complex, συμπληρωμένη

```
class complex {
private:
    double re, im;
public:
    complex(double, double);      // constructors
    complex(const complex &c);
    complex();
    ~complex();                   // destructor
    complex add(complex c);       // behaviour, exposed
    void print(ostream &out);
    void printN(ostream &out);
    double norm();
    double Re();                  // getters
    double Im();
    void setIm(double);           // setters
    void setRe(double);
    complex& operator=(const complex &y);   // overloaded
    bool operator>(complex c);
    friend ostream& operator<<(ostream &out, complex c);
};
```

δημιουργία

ανάθεση

σύγκριση

εκτύπωση

Class complex, συμπληρωμένη

```
// --- constructor with 2 parameters
complex::complex(double r, double i): re(r), im(i) {}

// --- constructor to copy another complex
complex::complex(const complex &c):
re(c.re), im(c.im) {}

// --- constructor to create a random complex
complex::complex() {
    // srand(time(NULL)) -> run once, not here!;
    // Dirty: returns a random between 0 and 99
    re = (double) (rand() % 100);
    im = (double) (rand() % 100);
}

// --- destructor
complex::~complex() {
    cout << "Dying: "; print(cout); } // for studying only
```

Class complex, συμπληρωμένη

```
complex complex::add(complex c) {
    return complex(re + c.re, im + c.im);
}

void complex::print(ostream &out) {
    out << re << "+" << im << "i   " ;
}

// prints norm, too"
void complex::printN(ostream &out) {
    out << re << "+" << im << "i  " <<
norm() << "   ";
}
```

Class complex, συμπληρωμένη

```
// --- getters
double complex::Re() {
     return re; }
double complex::Im() {
     return im; }

// --- setters
void complex::setRe(double rr) {
     re = rr; }
void complex::setIm(double ii) {
     im = ii; }
```

Class complex, συμπληρωμένη

```
// Assignment
complex& complex::operator=(const complex &y) {
    re = y.re;
    im = y.im;
    return *this;  // Τρέχον αντικείμενο!
}

// Comparison
bool complex::operator>(complex c) {
    return norm() > c.norm();
}

// Stream
ostream& operator<<(ostream &out, complex c) {
    out << "("<<c.re << ", " << c.im << "i)";
    return out;
}
```

Function templates (vi)

```
...
// needed by the constructor that
// generates random complex numbers
srand(time(NULL));

complex c[10];

cout << "Unsorted random complex numbers\n";
for (int i=0; i<10; ++i)
      c[i].printN(cout);

bubble_sort(10, c);

cout << "Sorted\n";
for (int i=0; i<10; ++i)
      c[i].printN(cout);
```

```
Unsorted random complex numbers
24+25i  34.6554
59+59i  83.4386
48+52i  70.7672
78+81i  112.45
22+19i  29.0689
13+1i  13.0384
74+25i  78.1089
12+52i  53.3667
49+33i  59.0762
91+63i  110.68
```

```
Sorted
13+1i  13.0384
22+19i  29.0689
24+25i  34.6554
12+52i  53.3667
49+33i  59.0762
48+52i  70.7672
74+25i  78.1089
59+59i  83.4386
91+63i  110.68
78+81i  112.45
```

Class templates (i)

- Τι κάνει η παρακάτω κλάση;

Στοίχιση!
Ονόματα!
Σχόλια!

```
class askisi2 {
private:
    string p;
    int s;
public:
    askisi2(const string &P, int d) :
        p(P), s(d) {}

    int get(const string &P) {
        if (P == p) return s;
        cout << "Nope!" << endl;
        exit(0);
    }
};
```

Class templates (i)

- Μυστικοί ακέραιοι αριθμοί

```
class secretInt {
private:
    string password;     // password
    int secretData;      // data element to hide
public:
    secretInt(const string &pwd, int d) :
        password(pwd), secretData(d) {}

    int get(const string &pwd) {
        if (pwd == password) return secretData;
        cout << "Wrong password!" << endl;
        exit(0);
    }
};
```

Class templates (ii)

- Μυστικές συμβολοσειρές

```
class secretStr {
private:
    string password;
    string secretData;
public:
    secretStr(const string &pwd, string d);
    string get(const string &pwd);
};
secretStr::secretStr(const string &pwd, string d):
  password(pwd), secretData(d) {}
string secretStr::get(const string &pwd) {
    if (pwd == password) return secretData;
    cout << "Wrong password!" << endl;
    exit(0);
}
```

Class templates (iii)

```
int main() {
  string secretPassw = "ntuaece4ever";
  string enteredPassw, secretString;
  int secretInteger;

  cout << "a secret int: "; cin >> secretInteger;
  cout << "a secret string: "; cin >> secretString;

  secretInt s1(secretPassw, secretInteger);
  secretStr s2(secretPassw, secretString);

  cout << "password: "; cin >> enteredPassw;
  cout << s1.get(enteredPassw) << endl;
  cout << s2.get(enteredPassw) << endl;
}
```

Class templates (iv)

- Μυστικό <ο,τιδήποτε>

```
template <typename T>
class secret {
private:
    string password;
    T secretData;
public:
    secret(const string &pwd, const T &d):
        password(pwd), secretData(d) {};

    T get(const string &pwd) {
        if (pwd == password) return secretData;
        cout << "Wrong password!" << endl;
        exit(0);
    }
};
```

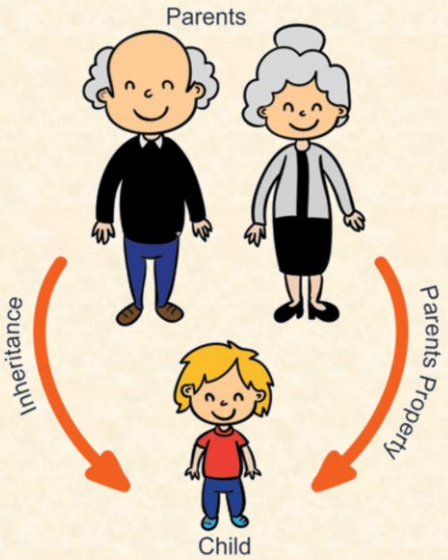
Class templates (v)

```
template <typename T>
class secret {
private:
    string password;
    T secretData;
public:
    secret(const string&, const T&);
    T get(const string&);
};

template <typename T>
secret<T>::secret(const string &pwd, const T &d):
    password(pwd), secretData(d) {};

template <typename T>
T secret<T>::get(const string &pwd) {
    if (pwd == password) return secretData;
    cout << "Wrong password!" << endl;
    exit(0); }
```

Ορισμός εκτός της κλάσης

Class templates (vi)

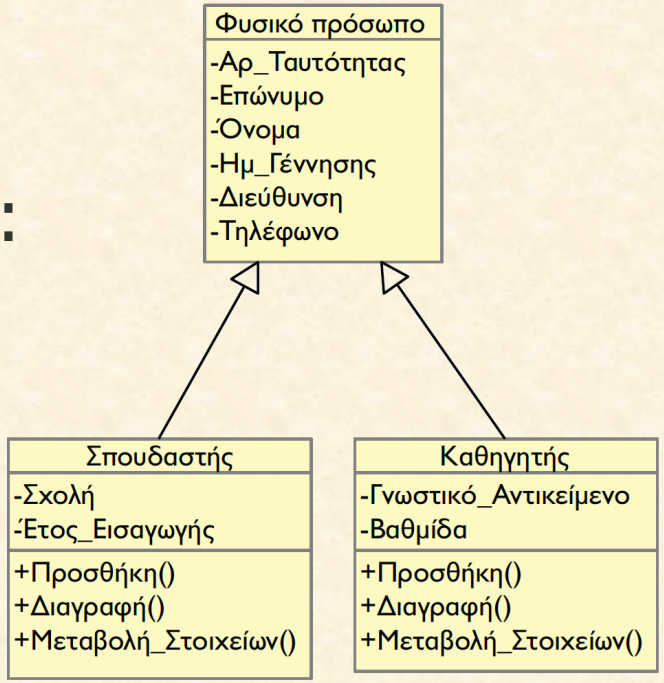
```
int main() {
  string secretPassw = "ntuaece4ever";
  string enteredPassw, secretString;
  int secretInteger;

  cout << "a secret int: "; cin >> secretInteger;
  cout << "a secret string: "; cin >> secretString;

  secret<int> s1(secretPassw, secretInteger);
  secret<string> s2(secretPassw, secretString);

  cout << "passwword: "; cin >> enteredPassw;
  cout << s1.get(enteredPassw) << endl;
  cout << s2.get(enteredPassw) << endl;
}
```



ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ

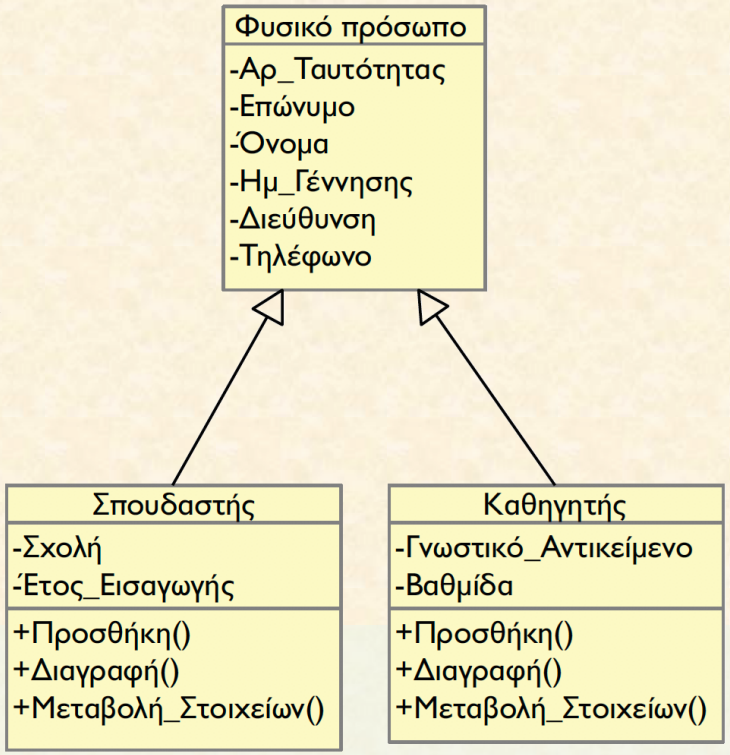
Κληρονομικότητα (inheritance)

- Η δυνατότητα ορισμού κλάσεων που αποκτούν (κληρονομούν) όλα τα κληροδοτούμενα χαρακτηριστικά μιας ή περισσοτέρων κλάσεων
  - Δεν κληροδοτούνται όλα τα μέλη μιας κλάσης
  - Απλή κληρονομικότητα:
    1 κλάση-γονέας
  - Πολλαπλή κληρονομικότητα:
    >1 κλάσεις-γονείς



Κληρονομικότητα

- Ισχυρό εργαλείο
  - Για επέκταση, προσθήκη, εξειδίκευση της δομής και συμπεριφοράς κλάσεων
  - Για επαναχρησιμοποίηση

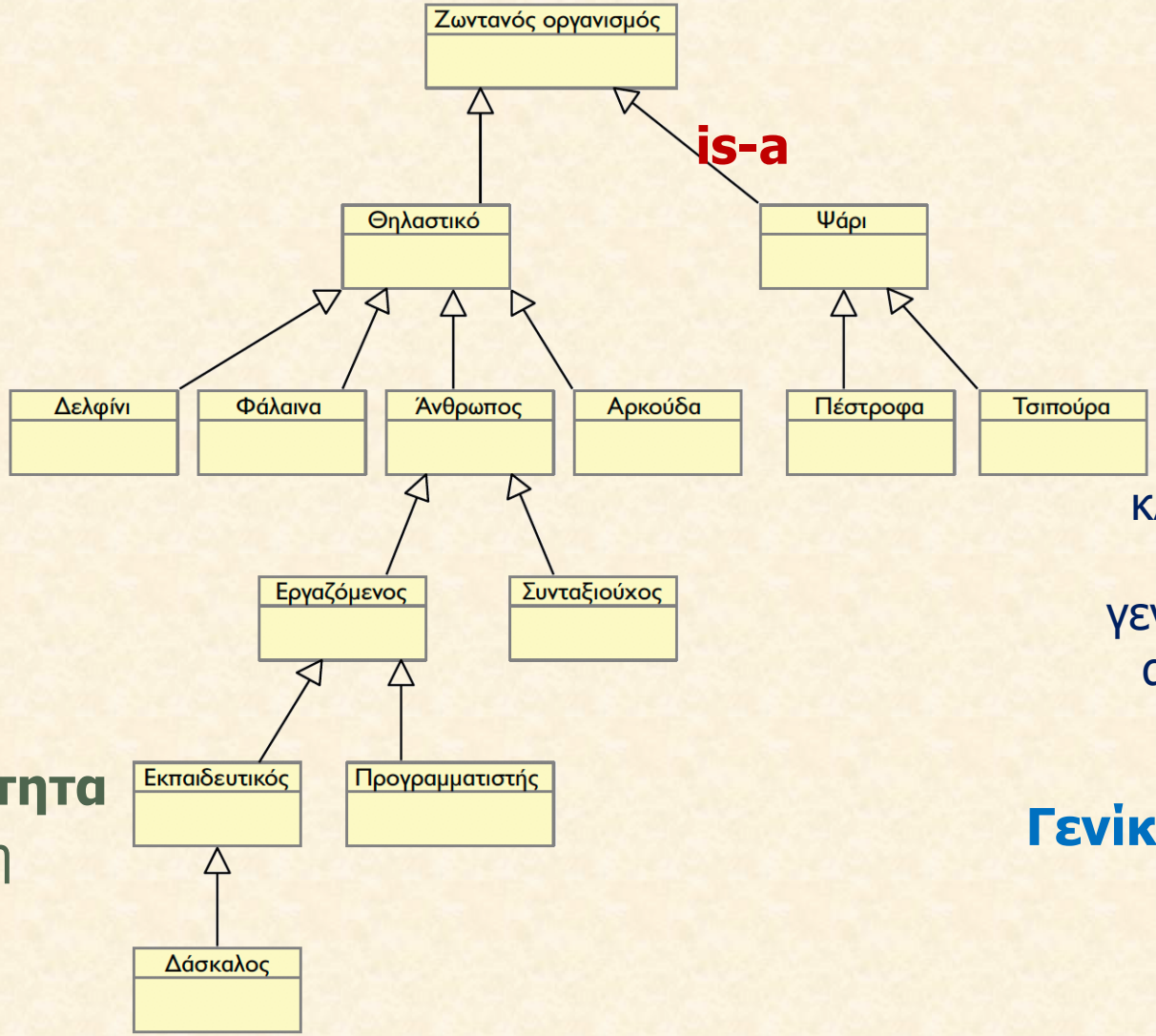


Υπενθύμιση!

Open/closed design principle

- A class should be open for extension and closed for modification.
- Classes should be written in a way that it is ready for adopting/adding new features but "not interested" in any modification.

Κληρονομικότητα και γενίκευση



Οι εξειδικευμένες κλάσεις κληρονομούν τα χαρακτηριστικά των γονέων τους

Κληρονομικότητα
Εξειδίκευση

Κάθε κλάση που κληρονομείται είναι γενικότερη από αυτή που την κληρονομεί

Γενίκευση (is-a)

Η κληρονομικότητα ως...

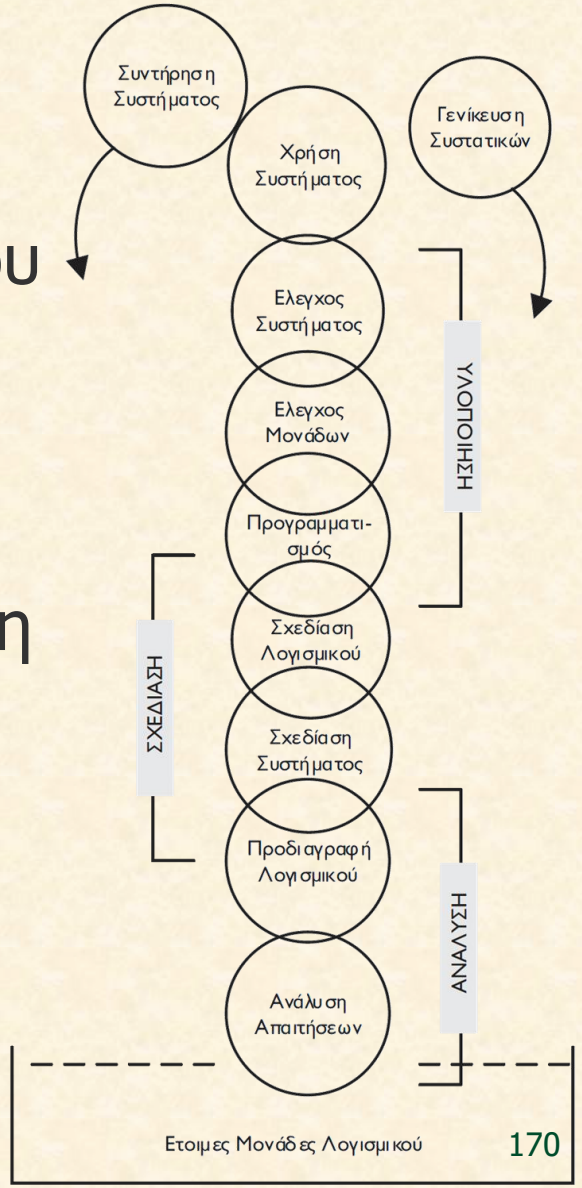
- ...μέσο μοντελοποίησης του πεδίου ενός προβλήματος:
  - Καλύτερη παράσταση ιδιοτήτων και ιεραρχιών ταξινόμησης
  - Δυνατότητα αποφυγής ή έγκαιρου εντοπισμού σφαλμάτων σχεδίασης
  - Αποφυγή "δημιουργικών" λύσεων οι οποίες θα έχουν επιπτώσεις αργότερα

Η κληρονομικότητα ως...

- ...προγραμματιστικό εργαλείο:
  - Επαναχρησιμοποίηση έτοιμου κώδικα δικού μας ή και τρίτων
  - Συγγραφή νέου κώδικα για επαναχρησιμοποιηση
  - Καλύτερη κατανόηση κώδικα
  - Παραγωγικότητα - καλύτερη ποιότητα κώδικα

Κληρονομικότητα

- Φιλοσοφία επαναχρησιμοποίησης πηγαίου κώδικα
  - Ενσωμάτωση συστατικών πηγαίου κώδικα, από μία "βιβλιοθήκη"
  - Εμπλουτισμός της "βιβλιοθήκης"
  - Συσσώρευση των διορθώσεων βελτιώσεων για μελλοντική χρήση



Κληρονομικότητα

- Επαναχρησιμοποίηση πηγαίου κώδικα
  - Οι δηλώσεις των πεδίων και των μεθόδων ξαναχρησιμοποιούνται, χωρίς να χρειάζεται να επαναληφθούν
  - Ο κώδικας μπορεί να έχει κατασκευαστεί σε κάποιο άλλο έργο

Υπενθύμιση!

Αρχή της μοναδικής ευθύνης (Single Responsibility Principle)

- Μια κλάση πρέπει να κάνει σωστά κάτι, για το οποίο να φέρει την αποκλειστική ευθύνη μέσα σε ένα πρόγραμμα
- Δηλαδή να μην επαναλαμβάνεται κώδικας, όταν αυτό μπορεί να αποφευχθεί

Κληρονομικότητα

- Πιο "σφιχτή" σχεδίαση
- Ορισμός κοινής διεπαφής (interface) που μοιράζονται περισσότερες κλάσεις
  - Μηχανισμός γενίκευσης
  - Χρήσιμο στους αφηρημένους τύπους δεδομένων (ΑΤΔ)
- Διεπαφή (interface)
  - "Συμβόλαιο" επικοινωνίας με τον έξω κόσμο
  - Μέθοδοι μέσω των οποίων η κλάση χρησιμοποιείται

Ορατότητα μελών και κληρονομικότητα

```
class [όνομα] {
```

public:

Μέλη ορατά εντός και εκτός της κλάσης

private: `// default!`

Μέλη ορατά μέσα στην κλάση και σε φίλες συναρτήσεις και κλάσεις

protected:

Μέλη ορατά μέσα στην κλάση, σε φίλες συναρτήσεις και κλάσεις και σε κλάσεις που τα κληρονομούν (κλάσεις-παιδιά)

```
};
```

Τύποι κληρονομικότητας

- **public**: Η κλάση-παιδί κληρονομεί τα public και protected μέλη της κλάσης-γονέα ως public και protected μέλη, αντίστοιχα, ισχύει το "is-a"
- **protected**: Η κλάση-παιδί κληρονομεί τα public και protected μέλη της κλάσης-γονέα ως protected μέλη, δεν ισχύει το "is-a"
- **private**: Η κλάση-παιδί κληρονομεί τα public και protected μέλη της κλάσης-γονέα ως private μέλη, δεν ισχύει το "is-a"

- Τα private μέλη ΔΕΝ κληρονομούνται

Υλοποίηση κληρονομικότητας

```
class parent_class
{
  public:
        ...
  protected:
        ...
  private:
        ...
};
```

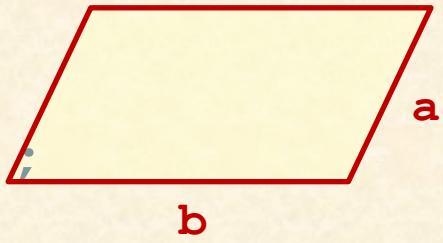
Προσοχή: Αφορά την κληρονομικότητα, όχι την ορατότητα μελών της κλάσης!

`public, protected, private`

```
class child_class : access_modifier parent_class
{
  public:
        ...
  protected:
        ...
  private:
        ...
};
```

Κληρονομικότητα, παράδειγμα

```
class myRectangle {
private:
    double a, b;
public:
    myRectangle();
    myRectangle(double, double);
    double A();
    double B();
    void setA(double);
    void setB(double);
    double perimeter();
    double area();
    void print(ostream&);
};
```



Κληρονομικότητα, παράδειγμα

```
class myRectangle {      // The parent-class
protected:
    double a, b;
public:
    myRectangle() : a(0), b(0) {}
    myRectangle(double A, double B) : a(A), b(B) {}
    double A() { return a; }
    double B() { return b; }
    void setA(double A) { a = A; }
    void setB(double B) { b = B; }
    double perimeter()    { return 2*(a+b); }
    double area()         { return a*b; }
    void print(ostream &out ) {
        out << a << " by " << b << ": a=" << area();
        out << " p=" << perimeter() << endl;
    }
};
```

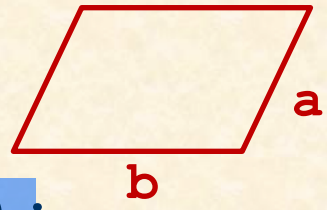
Κληρονομικότητα, παράδειγμα

```
int main() {
    myRectangle r;
    r.print(cout);
    myRectangle s(2,3);
    s.print(cout);
    r.setA(3);
    r.setB(4);
    r.print(cout);
}
```

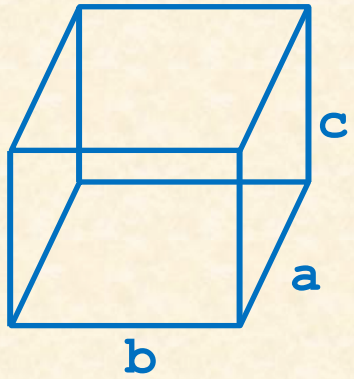
Κληρονομικότητα, παράδειγμα

re-use change new

```
class myCuboid {
private:
    double a, b, c;
public:
    myCuboid();
    myCuboid(double, double, double);
    double A(); double B(); double C();
    void setA(double);
    void setB(double);
    void setC(double);
    double perimeter();
    double aArea();
    double volume();
    void print(ostream&);
};
```



is-a



Κληρονομικότητα, παράδειγμα

```
class myRectangle {
protected:
     double a, b;
public:
     myRectangle() ;
     myRectangle(double,double);
     double A();
     double B();
     void setA(double);
     void setB(double);
     double perimeter();
     double area();
     void print(ostream&);
};
```

κληρονομημένα

επισκιάζοντα

επισκιασμένα

```
class myCuboid : public myRectangle {
protected:
     double c;
public:
     myCuboid();
     myCuboid(double,double,double);
     double perimeter();
     double area();
     double volume();
     double C();
     void setC(double);
     void print(ostream&);
};
```

Κληρονομικότητα - constructors

- Κατασκευαστές (constructors)

```
myRectangle::myRectangle() : a(0), b(0) {}
```

default constructor κλάσης-γονέα

```
myCuboid::myCuboid() : c(0) {}
```

default constructor κλάσης-παιδί

```
myCuboid c;
c.print(cout);    //a=0, b=0, c=0, area=0, perimeter=0, volume=0
```

Κληρονομικότητα - constructors

- Κατασκευαστές (constructors)

```
myRectangle::myRectangle(double A, double B) : a(A), b(B) {}
```

constructor κλάσης-γονέα

```
myCuboid::myCuboid(double A, double B, double C)
   : myRectangle(A, B), c(C) {}
```

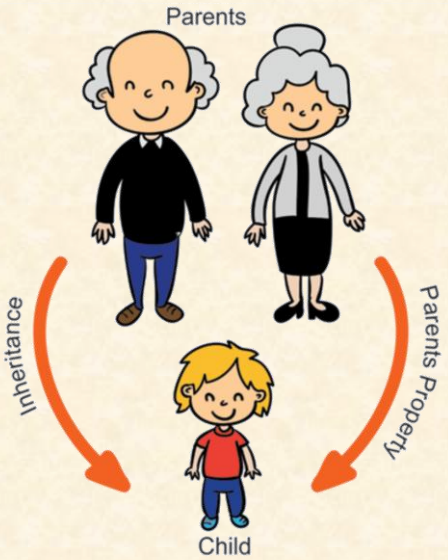
constructor κλάσης-παιδί

```
myCuboid c(1,2, 3);
c.print(cout);
//a=1, b=2, c=3, area=22, perimeter=24, volume=6
```

Κληρονομικότητα, παράδειγμα

Subtyping

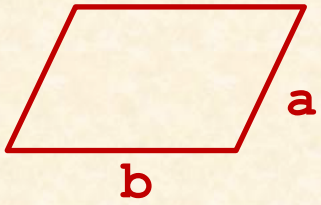
```
myCuboid c(2, 3, 4);
c.print(cout);

myRectangle *mr = &c;
mr->print(cout);            // τι τυπώνει;;;
```

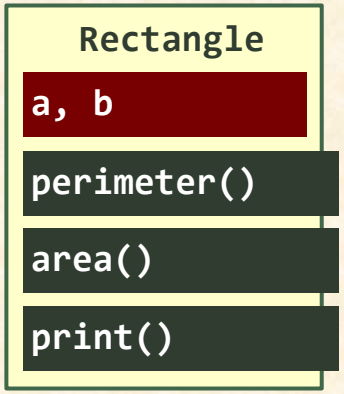


ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ (συνέχεια)

Πολλαπλή κληρονομικότητα

- Η κλάση Rectangle

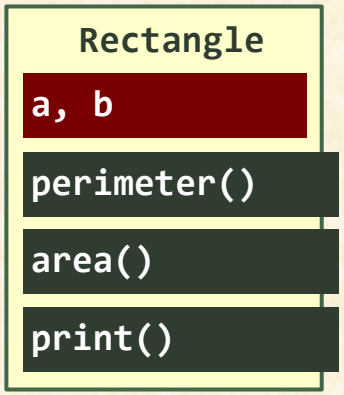





```
class Rectangle {
protected:
    double a, b;
public:
    Rectangle();
    Rectangle(double A, double B);

    double A();
    double B();
    void setA(double A);
    void setB(double B);

    double perimeter();
    double area();
    void print(ostream &out);

    friend ostream& operator<<(ostream &out, Rectangle r);
};
```

Πολλαπλή κληρονομικότητα

Η κλάση Rectangle

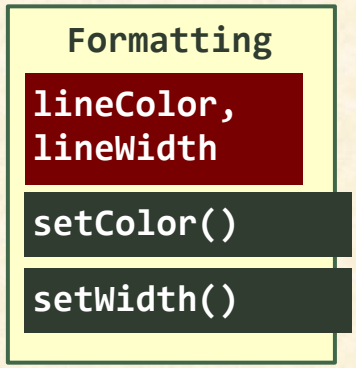



```
class Rectangle {      // The parent-class
protected:
    double a, b;
public:
  Rectangle() : a(0), b(0) {}
  Rectangle(double A, double B) : a(A), b(B) {}
  double A() { return a; }
  double B() { return b; }
  void setA(double A) { a = A; }
  void setB(double B) { b = B; }
  double perimeter() { return 2*(a+b); }
  double area() { return a*b; }
  void print(ostream &out) {out << "[a=" << a
        << " b=" << b << " a=" << area()
        << " p=" << perimeter() << "]\n";}
  friend ostream& operator<<(ostream &out, Rectangle r) {
    out << "[a=" << r.a << " b=" << r.b << " a="
        << r.area() << " p=" << r.perimeter() << "]";
    return out; }
};
```

```
int main (){
  Rectangle r(3, 10);
  r.print(cout);
  r.setA(15);
  r.print(cout);
}

[a=3 b=10 a=30 p=26]
[a=15 b=10 a=150 p=50]
```

Πολλαπλή κληρονομικότητα

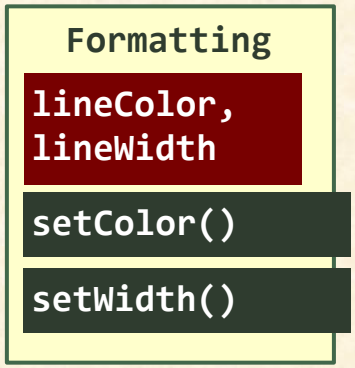
- Η κλάση Formatting




```
class Formatting {
protected:
  int lineWidth;
  string lineColor;
public:
  Formatting();
  Formatting(int W, string C);

  void setColor(string c);
  void setWidth(int w);
  friend ostream& operator<<(ostream &out, Formatting d);
};
```

Πολλαπλή κληρονομικότητα

Η κλάση Formatting



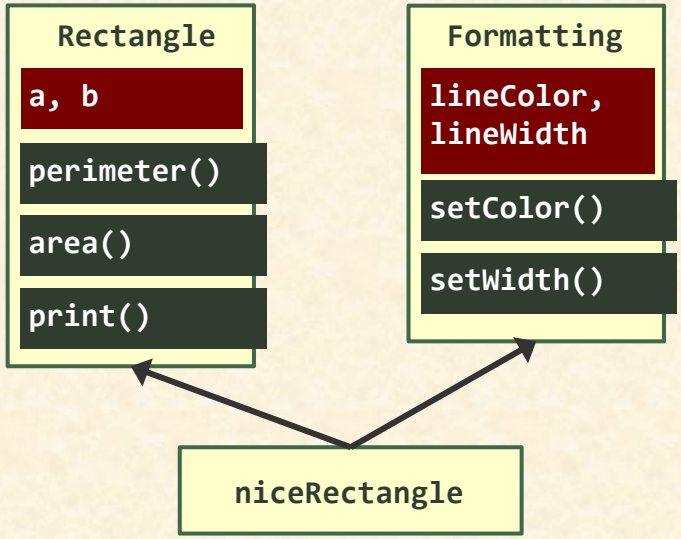

```
class Formatting {
protected:
  int lineWidth;
  string lineColor;
public:
  Formatting(): lineWidth(0), lineColor("transparent") {}
  Formatting(int W, string C): lineWidth(W), lineColor(C) {}

  void setColor(string c) { lineColor = c; }
  void setWidth(int w) { lineWidth = w; }

  friend ostream& operator<<(ostream &out, Formatting d) {
    out << "[color="<<d.lineColor<<" width="<< d.lineWidth<<"]";
    return out;
  }
};
```

```
int main (){
  Formatting f1(1,"blue");
  cout << f1 << endl;
  f1.setColor("red");
  cout << f1; }
```

```
[color=blue width=1]
[color=red width=1]
```

Πολλαπλή κληρονομικότητα

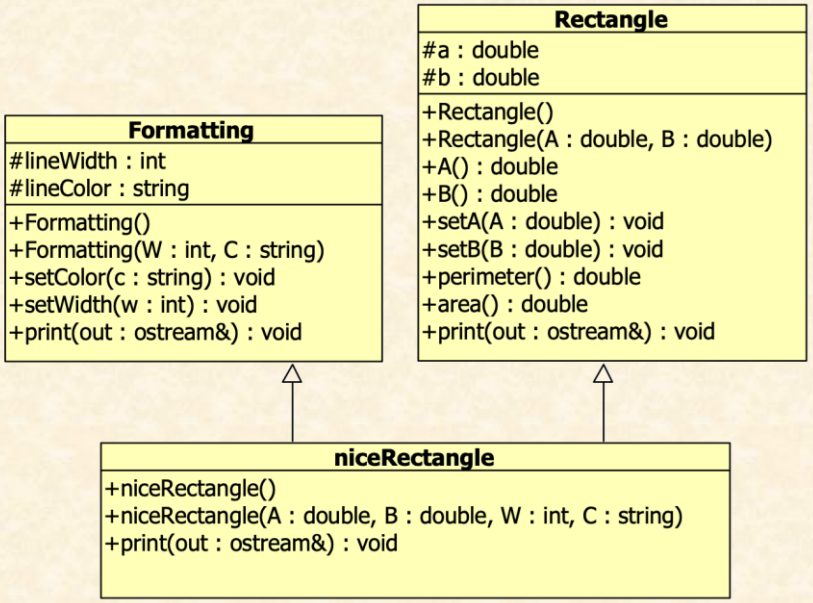
- Μια κλάση κληρονομεί από περισσότερες




```
class niceRectangle : public Formatting, public Rectangle {
public:
  niceRectangle();
  niceRectangle(double, double, int, string);
  void print(ostream &out );
  friend ostream& operator <<(ostream&, niceRectangle);
};
```

Πολλαπλή κληρονομικότητα

- Μια κλάση κληρονομεί από περισσότερες




```
class niceRectangle : public Formatting, public Rectangle {
public:
  niceRectangle();
  niceRectangle(double, double, int, string);
  void print(ostream &out );
  friend ostream& operator <<(ostream&, niceRectangle);
};
```

Πολλαπλή κληρονομικότητα

```
class niceRectangle : public Formatting, public Rectangle {
public:
  niceRectangle(): Rectangle(), Formatting() {}

  niceRectangle(double A, double B, int W, string C):
                Rectangle(A, B), Formatting(W, C) {}

  void print(ostream &out )  {
    out <<"(nr) "<< a << " by " << b << ": ar="
        << area() << ", per=" << perimeter()
        << ", linecol=" << lineColor << ", linewidth="
        << lineWidth << endl;
  }

  friend ostream& operator<<(ostream &out, niceRectangle NR) {
    NR.print(out); return out;
  }
};
```

Κλήση των constructors των κλάσεων - γονέων

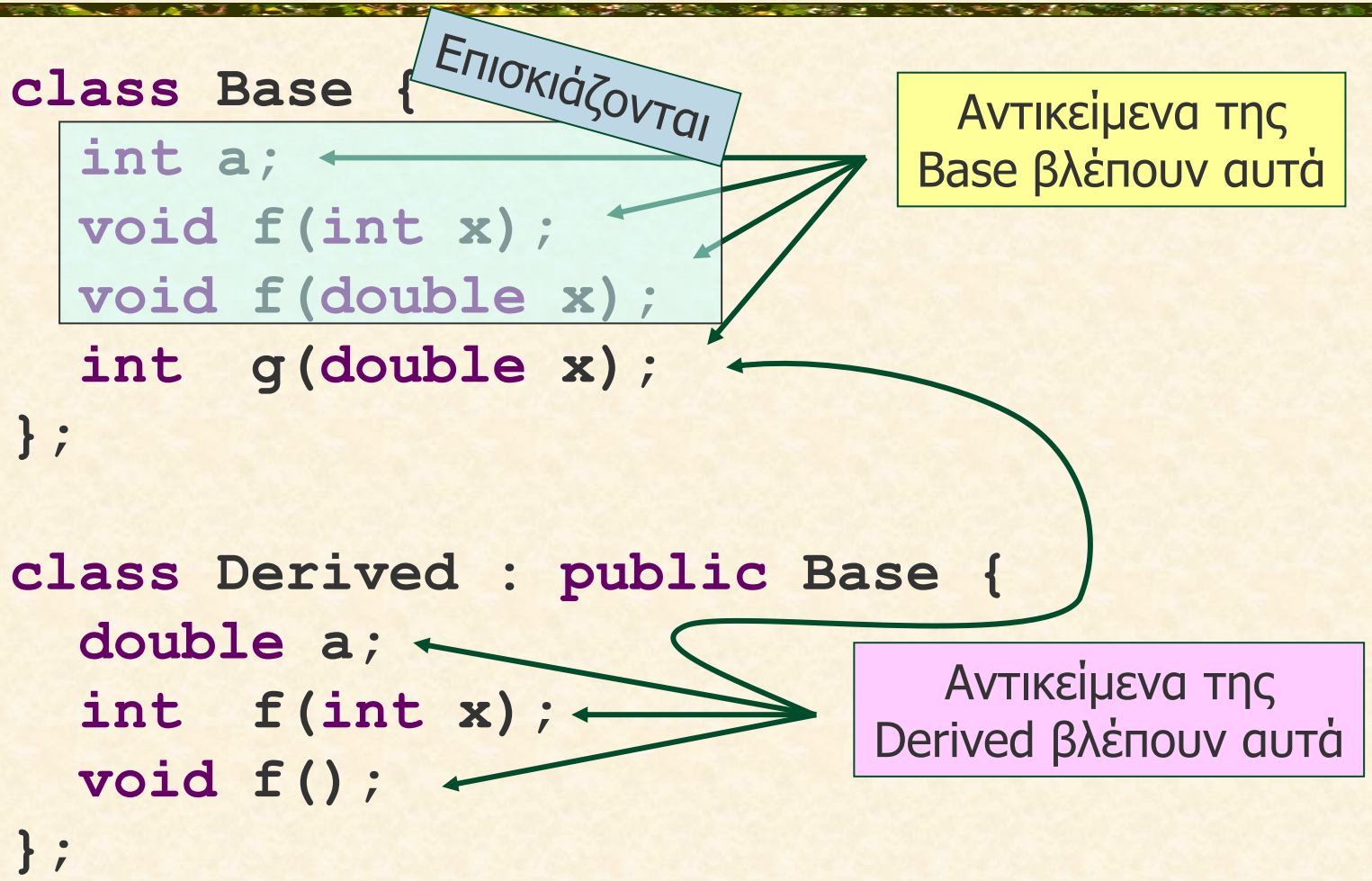
Από Rectangle

Από Formatting

Πολλαπλή κληρονομικότητα

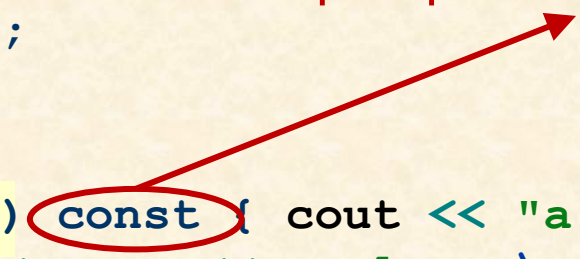
```
niceRectangle nr;
nr.print(cout); (nr) 0 by 0: ar=0, per=0, linecol=transparent, linewidth=0

niceRectangle nr2(2, 3, 1, "blue");
nr2.print(cout); (nr) 2 by 3: ar=6, per=10, linecol=blue, linewidth=1

cout<<"(nr2) area= "<<nr2.area()<<endl; (nr2) area= 6
nr2.setA(3);
cout<<"(nr2) per="<<nr2.perimeter()<<endl; (nr2) per=12

Rectangle *rr = &nr2;  // ? [a=3 b=3 ar=9 per=12]
rr->print();

Formatting *ff = &nr2; // ? [color=blue width=1]
ff->print();
```

Επισκίαση μεθόδων



Επισκίαση μεθόδων

```
class Person {
protected:
  string name;
public:
  ...
  void print() const { cout << "a person called "
            << name << " sleeps\n"; }
};

class Student : public Person {
private:
  string enrolledIn;
public:
  void print() const { cout << "a student called "
    << name << " reads about " << enrolledIn << "\n"; }
};
```



Επισκίαση μεθόδων

- Ολόκληρη η κλάση Person

```
class Person {

public:
    Person(): name("John Doe") {}                    // constructors
    Person(const string &newname): name(newname) {}

    string Name() { return name; }                   // getter

    void setName(const string &newname) { name = newname;} // setter

    virtual void print() const {
        cout << "person " << name << " sleeps\n";
    } // behaviour

protected:
    string name;
};
```

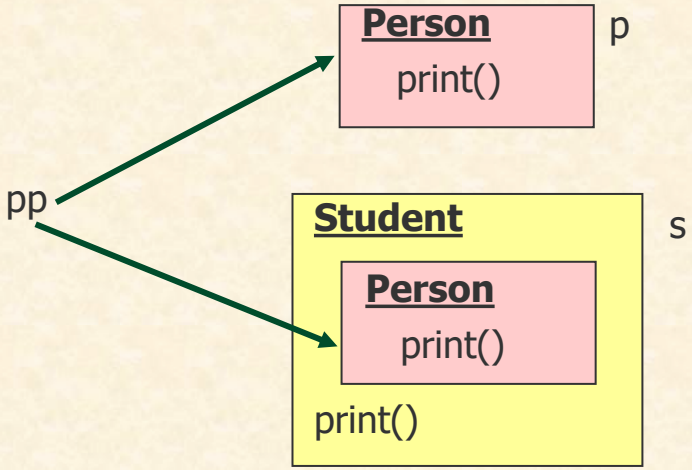
Επισκίαση μεθόδων

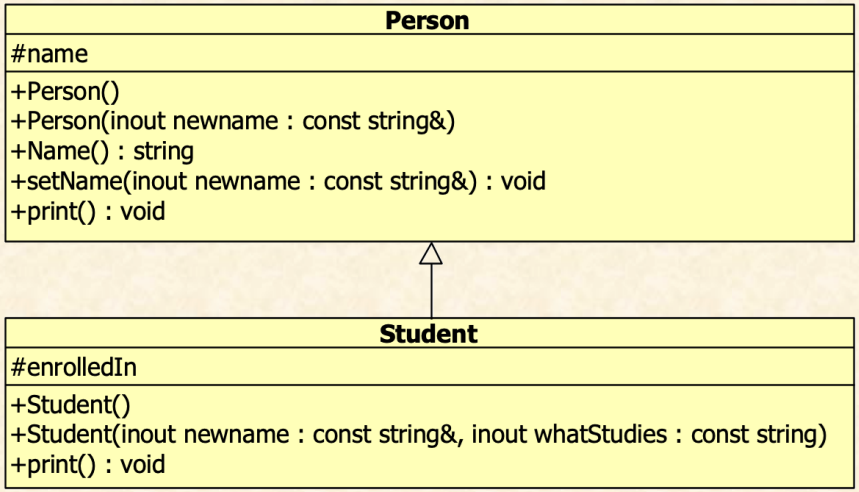
- Ολόκληρη η κλάση Student

```
class Student : public Person {

public:
    Student(): Person(), enrolledIn("maths") {}

    Student(const string &newname, const string whatStudies):
                        Person(newname), enrolledIn(whatStudies) {}

    void print() const override { cout << "student " << name
                            << " studies " << enrolledIn << "\n";
    }

protected:
    string enrolledIn;
};
```

Επισκίαση μεθόδων

```
Person p, *pp;
Student s;
p.print(); // person John Doe sleeps
s.print(); // student John Doe reads about maths

pp = &p;
pp->print(); // person John Doe sleeps
pp = &s;
pp->print(); // person John Doe sleeps
```





Εικονικές μέθοδοι

```
class Person {
protected:
  string name;
public:
  ...
  virtual void print() const {
    cout << "person " << name << " sleeps\n";}
};

class Student : public Person {
private:
  string enrolledIn;
public:
  void print() const override{ cout << "student "
    << name << " reads about " << enrolledIn << "\n"; }
};
```

Εικονικές μέθοδοι

```
Person p, *pp;
Student s;
p.print(); // person John Doe sleeps
s.print(); // student John Doe reads about maths

pp = &p;
pp->print(); // person John Doe sleeps

pp = &s;
pp->print();// student John Doe reads about maths
```