ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΤΕΧΝΙΚΕΣ

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

Διδάσκοντες:

Γιώργος Γκούμας
Άρης Παγουρτζής
Γιώργος Στάμου
Βασίλης Βεσκούκης
Θάνος Βουλόδημος
Γιώργος Αλεξανδρίδης
Γιώργος Σιόλας
Παρασκευή Τζούβελη

progtech@cslab.ece.ntua.gr

18/3/22

Διαφάνειες παρουσιάσεων

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

Σύντομη επανάληψη

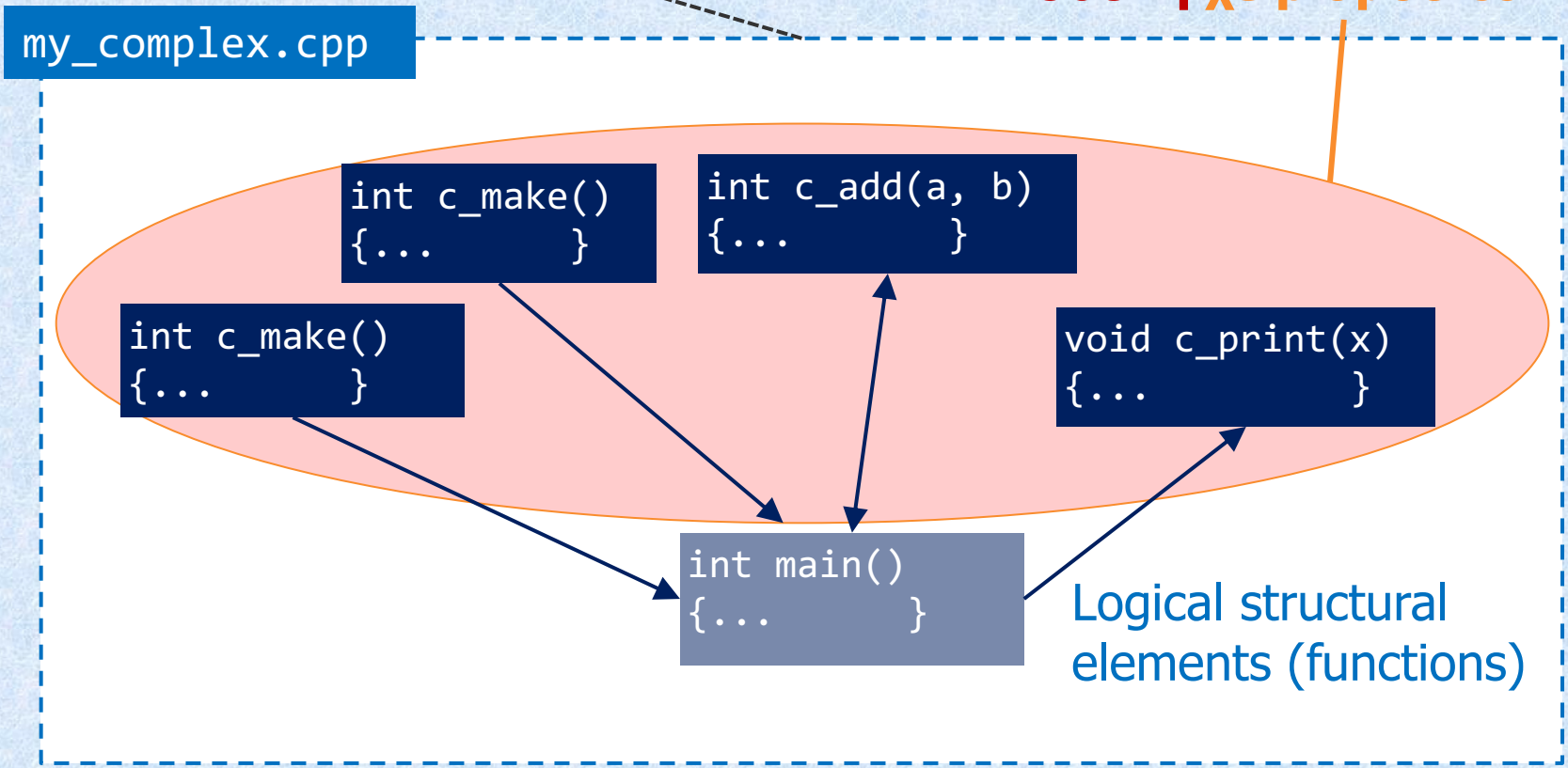
OBJECT-ORIENTED THINKING & PROGRAMMING

Εισαγωγή στο object-orientation

Δημιουργία, Διάθεση, Μεταβολή, Διαγραφή, Υλοποίηση πράξεων

Source code file (physical container)

Ευθύνη χειρισμού complex



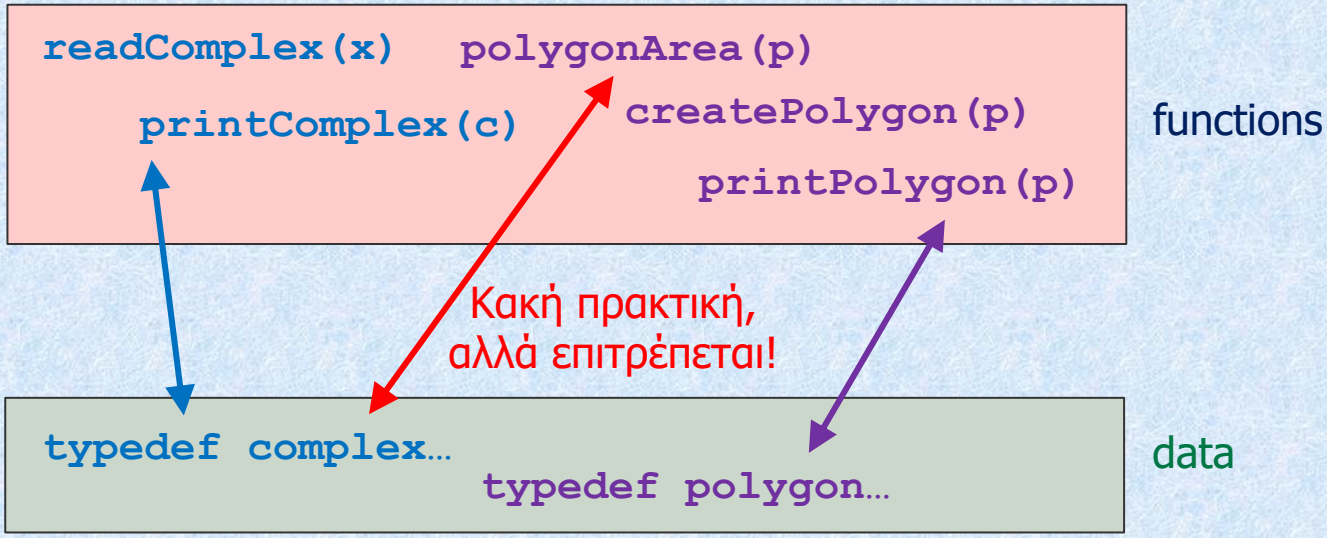
Εισαγωγή στο object-orientation

- Κλάσεις: οριζόμενες από τον χρήστη δομές, οι οποίες περιέχουν
  - Ιδιότητες (πεδία, fields, attributes)
  - Συμπεριφορά (μεθόδους, methods, member functions)

- Οι κλάσεις είναι:
  - Προγραμματιστικό εργαλείο
  - Εργαλείο μοντελοποίησης δεδομένων και της συμπεριφοράς τους

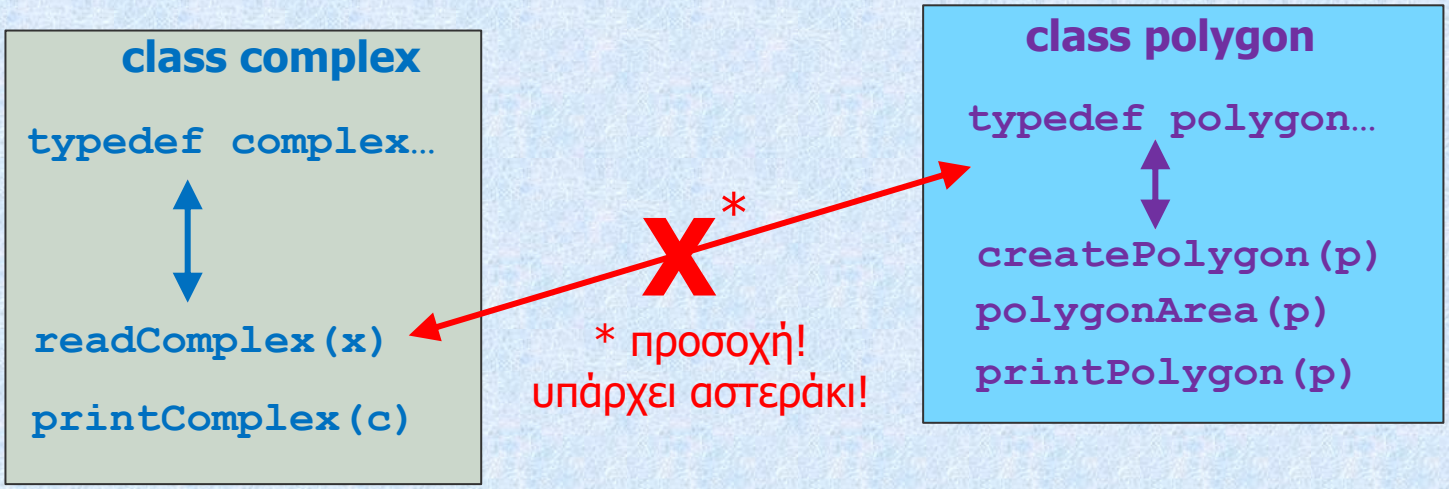
Shift of paradigm: χωρίς χρήση κλάσεων

- Ορισμός των δεδομένων από το πεδίο του προβλήματος ("εκφώνηση!")
- Ορισμός συναρτήσεων που δέχονται ως παραμέτρους τα δεδομένα που ορίστηκαν
- Χρήση των συναρτήσεων



Shift of paradigm: με χρήση κλάσεων

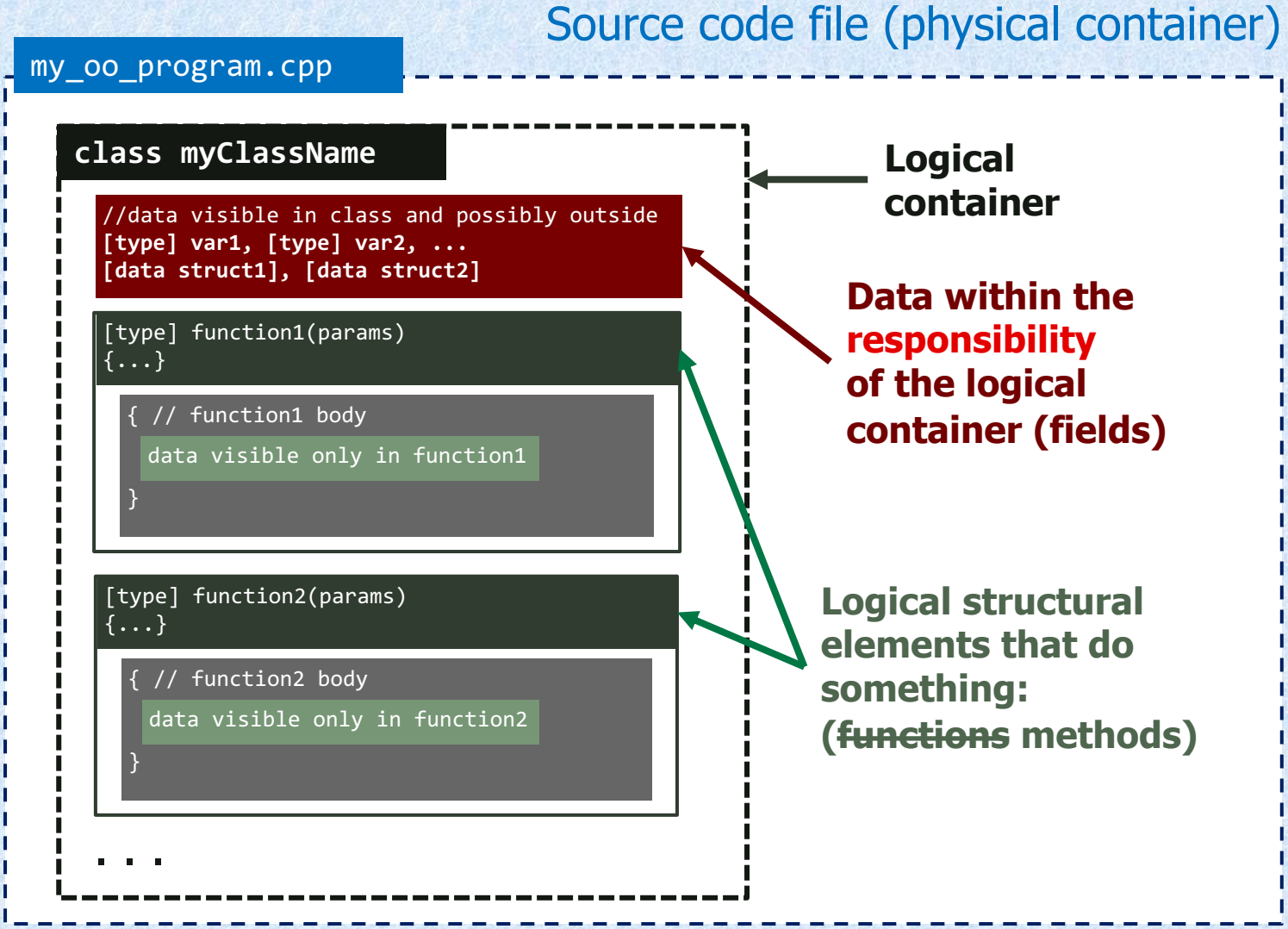
- **Data abstraction**:
  Ορισμός κάθε τύπου δεδομένων στο πεδίο του προβλήματος ως μία κλάση, που περιέχει
  - την περιγραφή των δεδομένων [πεδία]
  - τις συναρτήσεις που τα χειρίζονται [μέθοδοι]
- Δημιουργία και χρήση αντικειμένων



Κλάσεις - ορολογία

- Μη-αντικειμενοστρεφείς τύποι: **τύπος -> μεταβλητή**
  - **`int i`** [Μεταβλητή τύπου **`int`**]
  - **`Point p1`** [μεταβλητή τύπου **`Point`** (struct που ορίστηκε από τον προγραμματιστή και είναι νέος τύπος)]
- Κλάσεις: **Κλάση -> αντικείμενο (object)**
  - Όπως ορίζουμε μεταβλητές ενός τύπου, δημιουργούμε **αντικείμενα** μιας **κλάσης** που περιέχουν **δεδομένα (πεδία, κατάσταση)** και **συναρτήσεις (μεθόδους, συμπεριφορά)**
- Object-oriented programming = αντικειμενοστρεφής προγραμματισμός

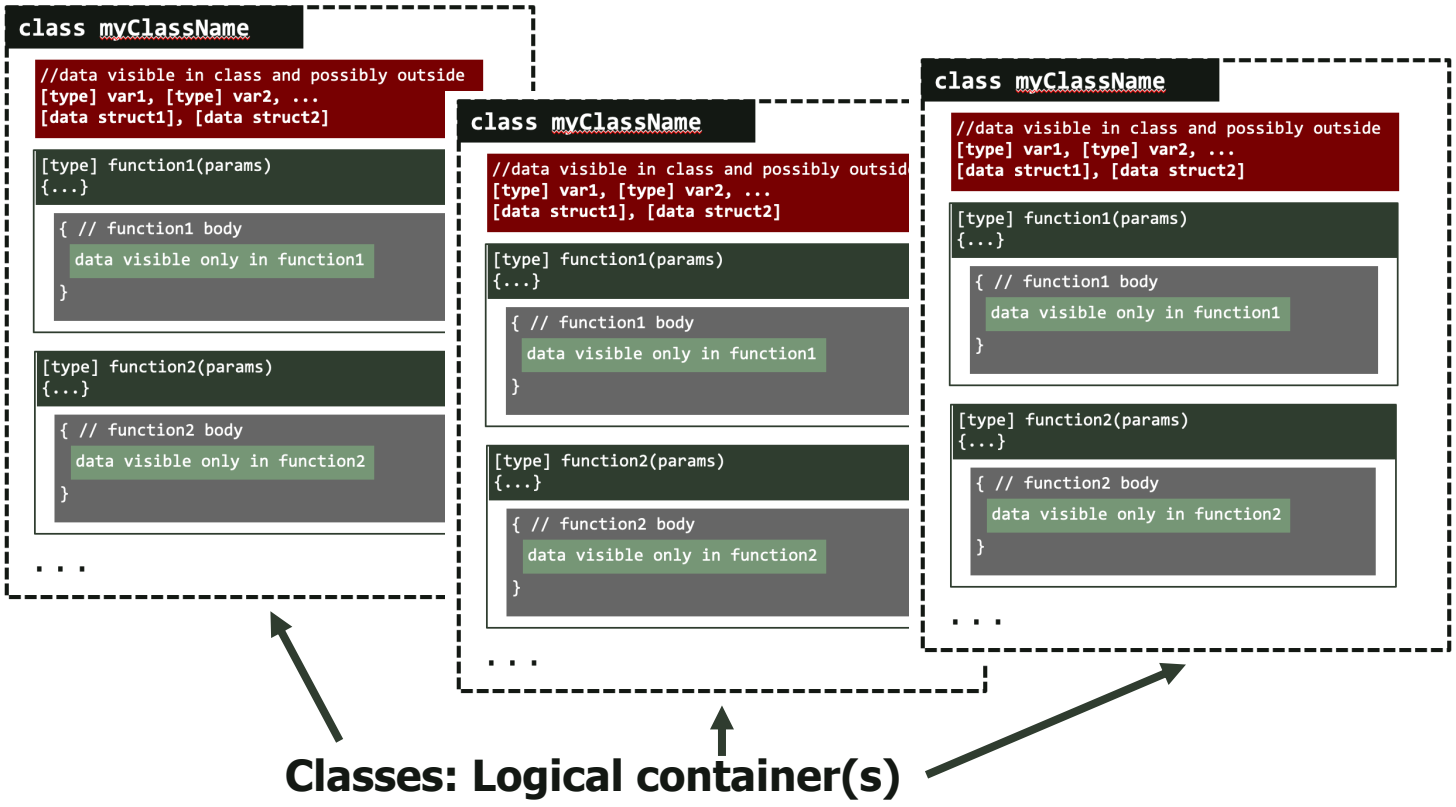
Εισαγωγή στο object-orientation



Εισαγωγή στο object-orientation

Source code file (physical container)

my_oo_program.cpp



Ορατότητα μελών κλάσης

```
class [όνομα] {
  public:
    Πεδία/μέθοδοι ορατά εντός και εκτός της κλάσης

  private: // default!
    Πεδία/μέθοδοι ορατά μόνο μέσα στην κλάση

  protected:
    Πεδία/μέθοδοι ορατά μέσα στην κλάση και στις
    κλάσεις-παιδιά
};
```

Θα επανέλθουμε, στη συζήτηση περί κληρονομικότητας

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

Κλάσεις… (iii)

- Οι κλάσεις σχεδιάζονται με διαφορετική φιλοσοφία (object-oriented)
  - αφαίρεση δεδομένων / απόκρυψη πληροφοριών
  - ενθυλάκωση (encapsulation) = πεδία και μέθοδοι μέσα στα αντικείμενα
  - περισσότερα στην πορεία
- Ακολουθεί παράδειγμα
  - Μαντέψτε: μιγαδικοί αριθμοί

Κατασκευαστές (constructors)

- Εκτελούνται αυτόματα με τη δήλωση ενός αντικειμένου
- Εχουν πάντα το όνομα της κλάσης
- ΔΕΝ έχουν τύπο
- Μπορεί να είναι περισσότεροι του ενός, αλλά πρέπει να έχουν διαφορετική λίστα παραμέτρων (signature)

Καταστροφείς (destructors) (i)

- Καλούνται όταν καταστρέφονται τα αντικείμενα (αποδεσμεύεται η μνήμη τους)
- Κάθε κλάση έχει μόνο έναν destructor

```
class complex {
public:
  ...
  ~complex() {
    cout << "a complex number dies... :-("
         << endl;
  }
  ...
};
```

Καλή πρακτική: μέθοδοι getters / setters

- Η ορατότητα των μελών μιας κλάσης από έξω πρέπει να είναι η εντελώς απαραίτητη:
  - Ορισμός αντικειμένων (constructors)
  - Απόδοση/ανάγνωση τιμών σε/από πεδία (setters/getters)
  - Υπηρεσίες της κλάσης προς τα έξω

- Με βάση αυτά (γενική "γραμμή"):
  - Όλα τα μέλη μιας κλάσης δηλώνονται private
  - Μέθοδοι getters, setters και υπηρεσίες δηλώνονται public

Καλή πρακτική: getters

- Μια μέθοδος "getter" επιστρέφει:
  - Την τιμή ενός πεδίου ή ένα αντικείμενο του τύπου του πεδίου
  - Το αποτέλεσμα ενός υπολογισμού (private μεθόδου της κλάσης) που εκτελείται εντός

- Καλή ιδέα:
  μια μέθοδος getter ΔΕΝ πρέπει να εμφανίζει output στην οθόνη

Καλή πρακτική: setters

- Μια μέθοδος "setter":
  - Συνήθως είναι τύπου void, εκτός αν πρέπει να επιστρέψει το αποτέλεσμα της εκτέλεσής της
  - Δίνει τις τιμές που δέχεται ως παράμετρους, στα αντίστοιχα πεδία της κλάσης
  - Προκαλεί την εκτέλεση όλων των μεθόδων που υπολογίζουν τιμές εξαρτημένων πεδίων (αν υπάρχουν), ώστε όλα τα πεδία να έχουν σωστές (=συνεπείς) τιμές

Παράδειγμα getters / setters

```
class line

  float x1,y1,x2,y2,len

  line(float X1,Y1,X2,Y2) {
    x1=X1; y1=Y1;
    x2=X2; y2=Y2;
    calclen();
  }

  void setX1(float X1) {
    x1 = X1;
    calclen(); }
  // same for setY1...

  float X1() {
    return x1; }
  float Y1() {
    return y1; }

  // same for X2,Y2,len...

  float calclen()
```

setter

getter

```
line myline(0,0,1,1);

cout << myline.LEN();  // 1.4142


myline.setX1(-1);
myline.setY1(-1);
cout << myline.LEN();  // 2.8284

cout << myline.X1() << "\n";
cout << myline.Y1() << "\n";
```

Η ευθύνη της συνέπειας των τιμών των πεδίων του αντικειμένου ανήκει στην κλάση!

Εισαγωγή στο object-orientation

- Υπερφόρτωση (overloading) τελεστών: + και <<

```
class complex {
public:
   complex(double r = 0.0, double i = 0.0);
   friend complex operator+(complex c1,
                            complex c2);
   friend ostream& operator<<(ostream &out,
                              complex c);
private:
   double re, im;
};
```

Εισαγωγή στο object-orientation

```
complex operator+(complex c1, complex c2) {
  return complex(c1.re + c2.re,
                 c1.im + c2.im);
}

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

int main() {
  complex c1(3, 4), c2(1, 2);
  complex c = c1 + c2;
  cout << c << endl;
}
```

Μέθοδοι και φίλες συναρτήσεις (iii)

- Και στις δύο περιπτώσεις η χρήση είναι ίδια: `c1 + c2`
- Στην περίπτωση #1 το αντικείμενο `c1` είναι αυτό για το οποίο καλείται η μέθοδος και το `c2` είναι η παράμετρος
- Στην περίπτωση #2 και τα δύο αντικείμενα είναι παράμετροι (πρόκειται για συνάρτηση, όχι για μέθοδο)

Μέθοδοι και φίλες συναρτήσεις (iv)

- Ο τελεστής εκτύπωσης όμως μπορεί να υλοποιηθεί μόνο ως φίλη συνάρτηση (γιατί;)

```
class complex {
public:
  ...
  friend ostream& operator<<(ostream &out,
                             const complex &x) {
    out << x.re << " + " << x.im << "i";
    return out;
  }
  ...
};
```

Περισσότεροι κατασκευαστές (i)

- Με χρήση υπερφόρτωσης
- Κοινός κατασκευαστής

```
complex(double r, double i) {
  re = r; im = i;
}
```

ή ισοδύναμα:

```
complex(double r, double i):
  re(r), im(i) {}
```

Η τελευταία μορφή διαχωρίζει την αρχικοποίηση των πεδίων του αντικειμένου από άλλες εντολές.

Περισσότεροι κατασκευαστές (ii)

- Default constructor

```
complex() { re = im = 0; }
```

ή ισοδύναμα:

```
complex(): re(0), im(0) {}
```

Κατασκευαστής χωρίς παραμέτρους, καλείται αυτόματα όταν έχουμε π.χ.

```
complex c;
```

Αν δεν οριστεί άλλος κατασκευαστής για μία κλάση, η C++ ορίζει αυτόματα έναν default

Περισσότεροι κατασκευαστές (iii)

- Copy constructor

```
complex(const complex &c) {
  re = c.re; im = c.im;
}
```

ή ισοδύναμα:

```
complex(const complex &c):
  re(c.re), im(c.im) {}
```

Καλείται αυτόματα όταν έχουμε π.χ.

```
complex c1(c);
complex c2 = c;
```

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;

```
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);
    }
};
```

Στοίχιση!
Ονόματα!
Σχόλια!

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);
    }
};
```

- 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 (ii)

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

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

  secretInt s1(secretPassw, secretInteger);
  cout << s1.get(enteredPassw);

  secretStr s2(secretPassw, secretString);
  cout << s2.get(enteredPassw);
}
```

Class templates (ii)

- 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);
    }
};
```

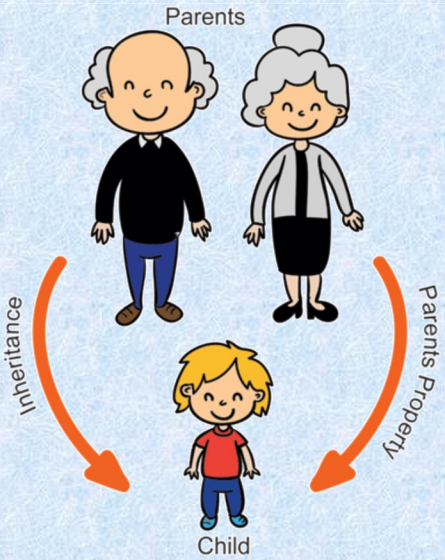
```
template <typename T>
class secret {
private:
    string password;
    T secretData;
public:
    secret(const string&, const T&);
    T get(const string&);
};

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

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

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

  secret<int> s1(secretPassw, secretInteger);
  cout << s1.get(enteredPassw);

  secret<string> s2(secretPassw, secretString);
  cout << s2.get(enteredPassw);
}
```

Class templates (ii)

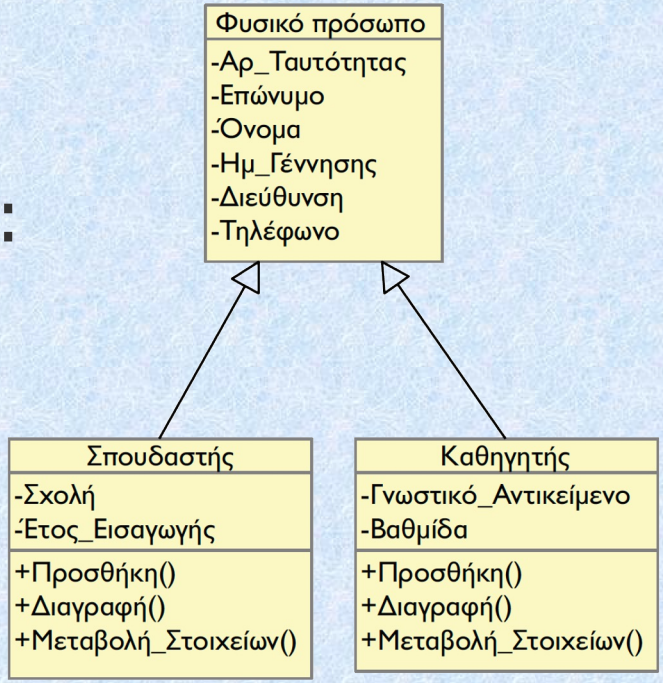
```
int main() {
  string secretPassw = "ntuaece4ever";
  string enteredPassw, secretString;
  int secretInteger;

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

  secretInt s1(secretPassw, secretInteger);
  cout << s1.get(enteredPassw);

  secretStr s2(secretPassw, secretString);
  cout << s2.get(enteredPassw);
}
```



ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ

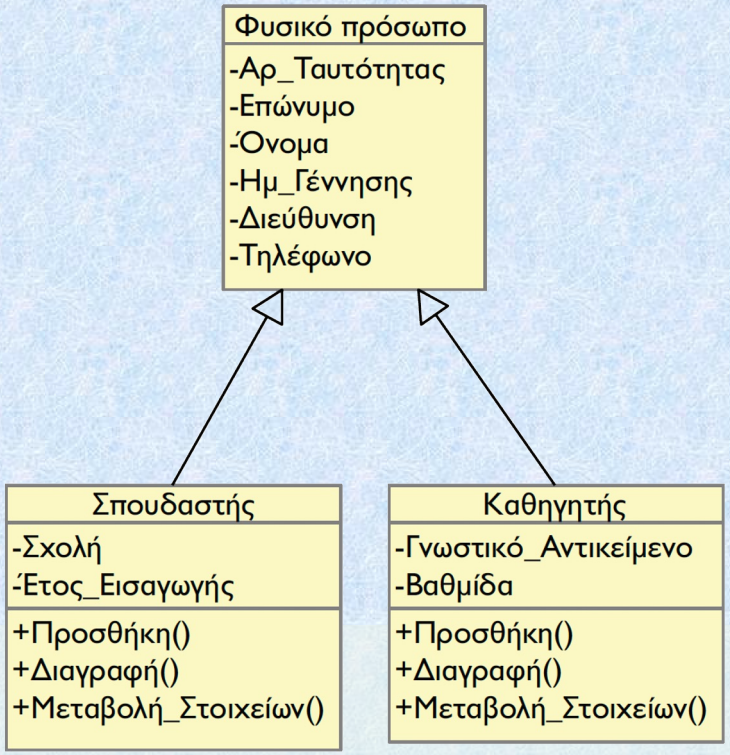
Κληρονομικότητα (inheritance)

- 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

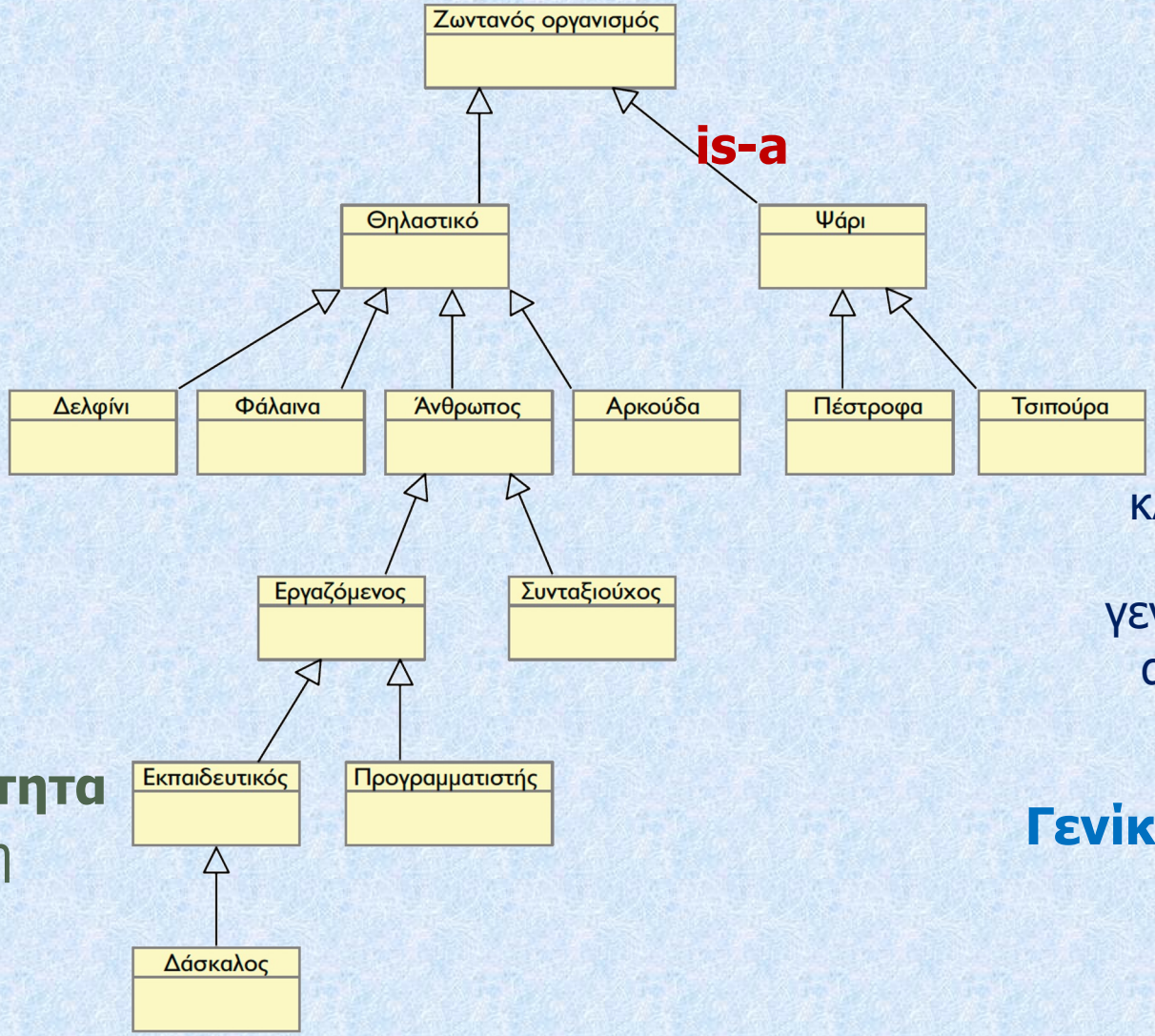


Υπενθύμιση!

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...

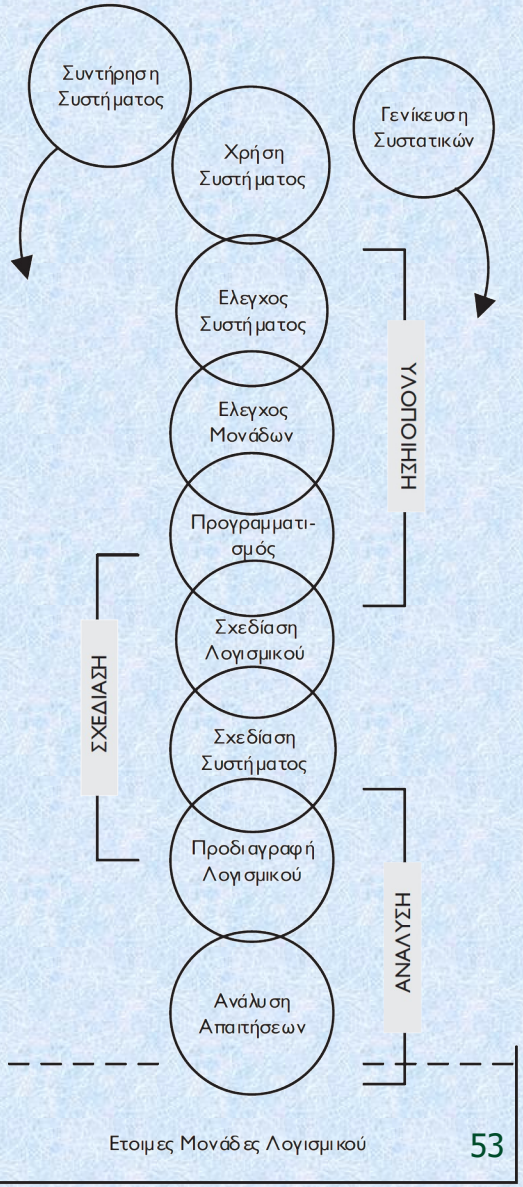
- ...μέσο μοντελοποίησης του πεδίου ενός προβλήματος:
  - Καλύτερη παράσταση ιδιοτήτων και ιεραρχιών ταξινόμησης
  - Δυνατότητα αποφυγής ή έγκαιρου εντοπισμού σφαλμάτων σχεδίασης
  - Αποφυγή "δημιουργικών" λύσεων οι οποίες θα έχουν επιπτώσεις αργότερα

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:
        ...
};
```

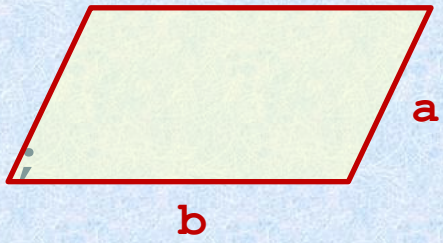
Αφορά την κληρονομικότητα, όχι την ορατότητα μελών της κλάσης!

`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;
    }
};
```

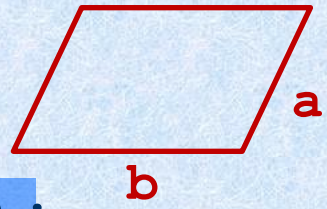
Κληρονομικότητα, παράδειγμα

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

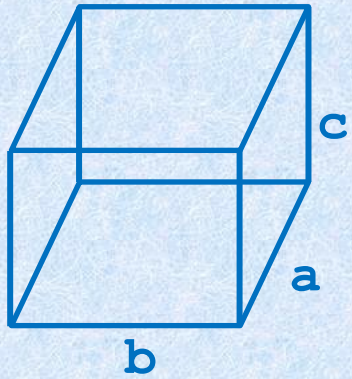
Κληρονομικότητα, παράδειγμα

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 A);
    void setB(double B);
    double perimeter();
    double area();
    void print(ostream &out );
};
```

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

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;;;
```