



Εισαγωγή στις κλάσεις – βασικές έννοιες

ΟΝΤΟΚΕΝΤΡΙΚΟΣ ΠΡΟΓΡ/ΣΜΟΣ C++

Μ. Ρήγκου (rigou@ceid.upatras.gr)

Άσκηση 1: Αναδρομή, rand()

anadromikh.cpp

- Γράψτε ένα πρόγραμμα που να ζητάει από το χρήστη να μαντέψει έναν αριθμό 1-5. Ο αριθμός που θα πρέπει να μαντέψει ο χρήστης θα πρέπει να δημιουργείται με τυχαίο τρόπο.
- `rand()`: συνάρτηση που παράγει τυχαίους αριθμούς σε ένα εύρος. Χρειάζεται να δηλώσουμε ένα `seed` καλώντας την `srand()` για να λειτουργήσει (συνήθως τρέχουσα ώρα συστήματος).

```
#include <cstdlib> // για τις synarthseis srand kai rand
```

```
#include <ctime> // για th synarthsh time
```

```
..... .
```

```
srand (time(0)); // time (0) δευτερόλεπτα που έχουν περάσει από την 1/1/1970  
// (UNIX epoch)!
```

```
v1 = rand() % 100; // ψευδοτυχαίοι αριθμοί από 0 έως 99
```

```
v2 = rand() % 30 + 1985; // ψευδοτυχαίοι αριθμοί από 1985 έως 2014
```

- Το πρόγραμμα θα ζητάει από το χρήστη αριθμούς επαναληπτικά μέχρι ο χρήστης να μαντέψει τον αριθμό (να υλοποιηθεί με αναδρομή)



Κλάσεις στη C++

- Οι κλάσεις στη C++ μοιάζουν με τις δομές με τη διαφορά ότι περιλαμβάνουν συναρτήσεις (**functions**) και έχουν προσδιοριστές πρόσβασης:
 - **(default) Private**: τα μέλη της κλάσης είναι προσβάσιμα μόνο από άλλα μέλη της ίδιας κλάσης (ή από *friends* της κλάσης).
 - **Protected**: τα μέλη της κλάσης είναι προσβάσιμα από άλλα μέλη της ίδιας κλάσης (ή από *friends* της κλάσης), αλλά και από κλάσεις απογόνους.
 - **Public**: τα μέλη είναι προσβάσιμα από παντού όπου είναι ορατό το αντικείμενο.

```
class Rectangle {  
    int width, height;  
    public:  
        void set_values (int,int);  
        int area (void);  
} ;
```

- Μια συνάρτηση **friend** μιας κλάσης ορίζεται έξω από την εμβέλεια της κλάσης αλλά έχει πρόσβαση σε όλες τις private και τις protected μεταβλητές της κλάσης.

Κλάσεις στη C++

```
// class example
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
public:
    void set_values (int,int);
    int area() {return width*height;}
};

void Rectangle::set_values (int x,
    int y) {
    width = x;
    height = y;
}

int main () {
    Rectangle rect;
    rect.set_values (3,4);
    cout << "area: " << rect.area();
    return 0;
}
```

```
// example: class constructor
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
public:
    Rectangle (int,int);
    int area () {return
(width*height);}
};

Rectangle::Rectangle (int a, int b) {
    width = a;
    height = b;
}

int main () {
    Rectangle rect (3,4);
    Rectangle rectb (5,6);
    cout << "rect area: " << rect.area()
<< endl;
    cout << "rectb area: " <<
rectb.area() << endl;
    return 0;
}
```

```

// classes and uniform initialization
#include <iostream>
using namespace std;

class Circle {
    double radius;
public:
    Circle(double r) { radius = r; }
    double circum() {return
        2*radius*3.14159265;}
};

int main () {
    Circle foo (10.0); // functional form
    Circle bar = 20.0; // assignment init.
    Circle baz {30.0}; // uniform init.

    cout << "foo's circumference: " <<
        foo.circum() << '\n';
    return 0;
}

```

Σε ένα κατασκευαστή η αρχικοποίηση τιμών μελών της κλάσης μπορεί να γίνει και **πριν το κυρίως σώμα του**:

```

//Member initialization in constructors

class Rectangle {
    int width,height;
public:
    Rectangle(int,int);
    int area() {return width*height;}
};

```

Μπορεί να οριστεί όλους τους παρακάτω τρόπους:

```

Rectangle::Rectangle (int x, int y) {
width=x; height=y;
}

Rectangle::Rectangle (int x, int y) : width(x) {
height=y;
}

Rectangle::Rectangle (int x, int y) : width(x),
height(y) {
}

```

Άσκηση 2: κλάσεις

- Γράψτε ένα πρόγραμμα στο οποίο να υλοποιείται μια κλάση για τη δημιουργία αντικειμένων τύπου “Μαθητής” (Student).
 - Τα χαρακτηριστικά ενός Student είναι τα:
 - Επώνυμο: Surname
 - Αριθμός μαθημάτων: num_of_courses
 - Μέσος όρος βαθμολογίας: av_grade
 - Αριθμός μητρώου: AM
 - Είδος μαθητή (Προπτυχιακός, Μεταπτυχιακός): kind
- Σημείωση: Για το χαρακτηριστικό «Είδος μαθητή» να χρησιμοποιηθεί **enum** με όνομα **Kind_of_Student** και τιμές **NOT_DEFINED,BSC,MSC**

```
enum Kind_of_Student { NOT_DEFINED,BSC,MSC};
```

```
class Student
```

```
{
```

```
    private: //idiotika dedomena
```

```
        string Surname;           //Epwnymo mathith
```

```
        int num_of_courses;       //Arithmos mathimatwn
```

```
        double av_grade;         //Mesos Oros
```

```
        int AM;                   //Arithmos Mhtrwoy
```

```
        Kind_of_Student kind;     //Eidos mathith
```

```
// .....ορισμός member functions.....
```

```
}
```

Άσκηση 2: κλάσεις

Να υλοποιηθούν οι συναρτήσεις – μέλη της κλάσης:

Συναρτήσεις Δημιουργίας (Constructors):

- ✓ **Student()** που αρχικοποιεί αυτόματα ένα αντικείμενο με τη δημιουργία του (προκαθορισμένες τιμές)
- ✓ **Student(string n,int c,double ag,int am, Kind_of_Student k)** που αρχικοποιεί ένα αντικείμενο με τη δημιουργία του, με ορίσματα που δίνονται κατά την κλήση.

Η συνάρτηση αποδιάρθρωσης (Destructor):

- ✓ **~Student()**


```
public:
    Student()
    {
        Surname="NoSurname";
        num_of_courses=0;
        av_grade=0;
        AM=0;
        kind=NOT_DEFINED;
    }
    Student(string n,int c,double ag,int am, Kind_of_Student k)
    {
        Surname=n;
        num_of_courses=c;
        av_grade=ag;
        AM=am;
        kind=k;
    }

    ~Student()
    {
        cout<<"destructor runs for:"<<endl;
        cout<<"Student "<<endl;
        print_Student(); // ο ορισμός της ακολουθεί
        system("PAUSE");
    }
}
```

Άσκηση 2: κλάσεις

Οι συναρτήσεις set και get:

- ✓ **set_Student(string n, int c, double ag, int am, Kind_of_Student k)** που εκχωρεί τιμές στα δεδομένα ενός αντικειμένου τύπου Student.
- ✓ **set_Surname(string n), set_num_of_courses(int c), set_av_grade(double ag), set_AM(int am), set_kind(Kind_of_Student k)** που εκχωρούν (set δηλ γράφουν) τιμές στα δεδομένα ενός αντικειμένου τύπου Student, Surname, num_of_courses, av_grade, AM, και kind αντίστοιχα.
- ✓ **get_Surname(), get_num_of_courses(), get_av_grade(), get_AM(), get_kind()** που επιστρέφουν (get δηλ διαβάζουν) τις τιμές των δεδομένων ενός αντικειμένου τύπου Student, Surname, num_of_courses, av_grade, AM, και kind αντίστοιχα.

Η συναρτήση:

- ✓ **print_Student()** που τυπώνει στην οθόνη όλα τα δεδομένα ενός αντικειμένου τύπου Student.

Getters και Setters

```
void set_Surname(string n)
    {   Surname=n;   }

void set_num_of_courses(int c)
    {   num_of_courses=c;
}

void set_av_grade(double ag)
    {   av_grade=ag;   }

void set_AM(int am)
    {   AM=am;   }

void set_kind(Kind_of_Student
k)
    {   kind=k;   }
```

```
string get_Surname()
    {   return Surname;   }

int get_num_of_courses()
    {   return num_of_courses;   }

double get_av_grade()
    {   return av_grade;   }

int get_AM()
    {   return AM;   }

Kind_of_Student get_kind()
    {   return kind;   }
```

Print_Student()

```
void print_Student()
{
    cout<<"Surname: "<<Surname<<endl;
    cout<<"AM: "<<AM<<endl;
    cout<<"Average grade: "<<av_grade<<" in
"<<num_of_courses<<" courses."<<endl;
    cout<<"Type of student: ";
    switch(kind)
    {
        case NOT_DEFINED:
            cout<<"Not defined"<<endl;
            break;
        case BSC:
            cout<<"BSc Student"<<endl;
            break;
        case MSC:
            cout<<"MSc Student"<<endl;
            break;
    }
}
```

Άσκηση 2: κλάσεις

Οι συναρτήσεις :

- ✓ **insert Stud Surname(), insert Stud AM(), insert Stud av_grade(), insert Stud num_of_courses(), insert Stud kind()** που για τα χαρακτηριστικά ενός αντικειμένου τύπου Student, Surname, AM, av_grade, num_of_courses, και kind αντίστοιχα κάνουν τα εξής:
 - Τυπώνουν μήνυμα στην οθόνη και ζητούν από το χρήστη να εισάγει το αντίστοιχο χαρακτηριστικό.
 - Πρέπει να ελέγχουν αν οι τιμές που δίνει ο χρήστης (εκτός από το επώνυμο) είναι έγκυρες και αν δεν είναι, να τις ξαναζητούν μέχρι ο χρήστης να δώσει σωστή τιμή (**χρησιμοποιήστε αναδρομή**).

Περιορισμοί που πρέπει να ελεγχθούν:

Πρέπει $AM > 0$, $stud_av_grade \geq 0$ και $stud_av_grade \leq 10$, $stud_num_of_courses > 0$ και ότι το kind θα πάρει valid τιμή.

- Καταχωρούν το χαρακτηριστικό που δόθηκε με την βοήθεια της αντίστοιχης συνάρτησης set.

```
void insert_stud_Surname()    {
    string sur;
    cout<<"Give student Surname:"<<endl;
    cin>>sur;
    set_Surname(sur);
}

void insert_stud_AM()        {
    int AM_c;
    cout<<"Give student AM:"<<endl;
    cin>>AM_c;
    if(AM_c>0)
        set_AM(AM_c);
    else
    {
        cout<<"Not valid AM - ";
        insert_stud_AM();
    }
}

void insert_stud_av_grade() {
    double av_grade_c;
    cout<<"Give student Average Grade:"<<endl;
    cin>>av_grade_c;
    if((av_grade_c>=0) && (av_grade_c<=10.00))
        set_av_grade(av_grade_c);
    else
    {
        cout<<"Not valid Average Grade - ";
        insert_stud_av_grade();
    }
}
```

```
void insert_stud_kind()    {
    int kind_c;
    cout<<"Give student kind "<<endl
    << "(0 for Not defined,"<<endl
    << "(1 for BSc Students,"<<endl
    << "(2 for MSc Students)"<<endl;

    cin>>kind_c;
    if(kind_c==0)
        set_kind(NOT_DEFINED);
    else if(kind_c==1)
        set_kind(MSC);
    else if(kind_c==2)
        set_kind(BSC);
    else
    {
        cout<<"Not valid kind - ";
        insert_stud_kind();
    }
}
```

Άσκηση 2: κλάσεις

Η συνάρτηση:

- ✓ **void insert_stud_info()** που χρησιμοποιώντας τις συναρτήσεις insert_stud_Surname(), insert_stud_AM(), insert_stud_av_grade(), insert_stud_num_of_courses(), insert_stud_kind() ζητάει από το χρήστη να δώσει τιμές στα χαρακτηριστικά ενός αντικειμένου τύπου Student.

```
void insert_stud_info()
{
    insert_stud_Surname();
    cout<<endl;
    insert_stud_AM();
    cout<<endl;
    insert_stud_av_grade();
    cout<<endl;
    insert_stud_num_of_courses();
    cout<<endl;
    insert_stud_kind();
    cout<<endl;
}
```


Άσκηση 2: κλάσεις

Να γραφτεί κώδικας στη main για τα εξής:

- Δημιουργία ενός αντικειμένου τύπου Student με όνομα "s1"
- ✓ Εκτύπωση τιμών δεδομένων του αντικειμένου με χρήση της συνάρτησης `print_Student()`
- ✓ Αλλαγή των τιμών των δεδομένων του με νέες τιμές που θα δώσει ο χρήστης, με χρήση της συνάρτησης `insert_stud_info()`
- ✓ Εκτύπωση τιμών δεδομένων του αντικειμένου με χρήση της συνάρτησης `print_Student()`



Συνάρτηση main()

```
int main()
{
    Student s1;

    s1.print_Student();
    cout<<endl;

    s1.insert_stud_info();
    cout<<endl;

    s1.print_Student();
    cout<<endl;
    system("pause");           // anamoni systhmatos
    return 0;                  // deixnei epityxh termatismo
}
```

Ο πλήρης κώδικας της
άσκησης 2 είναι στο αρχείο
class1.cpp

Άσκηση 3: κλάσεις και πίνακες

- Χρησιμοποιήστε την κλάση `Student` που δημιουργήθηκε στην Άσκηση 2 (αρχείο `class1.cpp`).

Να υλοποιηθούν οι συναρτήσεις:

- **`Fill_St_Arr(Student arr[],int size)`** που παίρνει ως όρισμα έναν πίνακα με στοιχεία τύπου `Student` και το μέγεθος του και ζητάει από το χρήστη να εισάγει στοιχεία για κάθε `Student` του πίνακα. Για την ανάθεση χρησιμοποιείτε τη συνάρτηση – μέλος της κλάσης `Student`, `insert_stud_info()`.
- **`Print_St_Arr(Student arr[],int size)`** που παίρνει ως όρισμα έναν πίνακα με στοιχεία τύπου `Student` και το μέγεθος του και τυπώνει όλα τα στοιχεία του. Για τις εκτυπώσεις χρησιμοποιείτε τη συνάρτηση – μέλος της κλάσης `Student`, `print_Student()`.

```
void Fill_St_Arr(Student arr[],int size)
{
    int i;
    cout<<"---Give students info:---"<<endl;
    for(int i=0;i<size;i++)
    {
        cout<<"Student "<<i+1<<":"<<endl;
        arr[i].insert_stud_info();
    }
}
```

```
void Print_St_Arr(Student arr[],int size)
{
    int i;
    cout<<"---Students info:---"<<endl;
    for(int i=0;i<size;i++)
    {
        cout<<"Student "<<i+1<<":"<<endl;
        arr[i].print_Student();
        cout<<endl;
    }
}
```

Άσκηση 3: κλάσεις και πίνακες

Να υλοποιηθεί η συνάρτηση:

Axiologhsh (Student arr[], int size) που παίρνει ως όρισμα έναν πίνακα με στοιχεία τύπου Student και το μέγεθος του και έχει σκοπό την αξιολόγηση του συνόλου των φοιτητών. Γράψτε κώδικα έτσι ώστε η συνάρτηση αυτή να τυπώνει :

- ✓ Τον αριθμό των προπτυχιακών φοιτητών.
- ✓ Τον αριθμό των μεταπτυχιακών φοιτητών.
- ✓ Τον αριθμό των φοιτητών που δεν έχει οριστεί το είδος τους (με kind== NOT_DEFINED).
- ✓ Το αποτέλεσμα της αξιολόγησης, δηλαδή το ποια είναι η ομάδα φοιτητών (μεταπτυχιακοί ή προπτυχιακοί) με το μεγαλύτερο μέσο όρο βαθμολογίας και το μέσο όρο αυτό.

```

void Axiologhsh(Student arr[],int size)
{
    int i; //metavlhth poy tha xrhsimopoihthei mesa stis for
    double sum_msc=0; //metavlhth gia to athrisma twv vathmwv twv MSC mathitwv
    double sum_bsc=0; //metavlhth gia to athrisma twv vathmwv twv BSC mathitwv
    double mo_msc=0; //metavlhth gia ton ypologismo toy mesoy oroy twv MSC mathitwv
    double mo_bsc=0; //metavlhth gia ton ypologismo toy mesoy oroy twv BSC mathitwv
    int msc_counter=0; //metavlhth gia thn katametrhsh twv MSC mathitwv
    int bsc_counter=0; //metavlhth gia thn katametrhsh twv BSC mathitwv
    int unknown_counter=0; //metavlhth gia thn katametrhsh twv mathitwv me agnwsto
    typo

    cout<<"---Students info:---"<<endl;
    for(int i=0;i<size;i++)
    {
        if(arr[i].get_kind()==BSC
        {
            sum_bsc+=arr[i].get_av_grade();
            bsc_counter++;
        }
        else if(arr[i].get_kind()==MSC)
        {
            sum_msc+=arr[i].get_av_grade();
            msc_counter++;
        }
        else
        {
            unknown_counter++;
        }
    }
    mo_bsc=sum_bsc/bsc_counter;
    mo_msc=sum_msc/msc_counter;
}

```

ΣΥΝΕΧΙΖΕΤΑΙ

```
// ...συνέχεια συνάρτησης Axiologhsh()

cout<<"Axiologhsh foithwn:"<<endl;
cout<<"Aritmos proptyxiakwn:"<<bsc_counter<<endl;
cout<<"Aritmos metaptyxiakwn:"<<msc_counter<<endl;
cout<<"Aritmos foithwn poy den symmethxan sthn katametrhsh:"<<unknown_counter<<endl;

cout<<endl<<"Apotelesma:"<<endl;
if(mo_bsc>mo_msc)
cout<<"Oi proptyxiakoi foithtes exoyn megalytero meso oro:"<<mo_bsc<<endl;
else if(mo_bsc<mo_msc)
cout<<"Oi metaptyxiakoi foithtes exoyn megalytero meso oro:"<<mo_msc<<endl;
else
cout<<"Metaptyxiakoi kai proptyxiakoi foithtes exoyn idio meso oro:"<<mo_msc<<endl;

}          // τέλος συνάρτησης Axiologhsh()
```

Άσκηση 3: κλάσεις και πίνακες

Να γραφτεί κώδικας στη main για τα εξής:

- ✓ Δημιουργία ενός πίνακα αντικειμένων τύπου `Student` με όνομα `s_array` και μέγεθος 4.
- ✓ Γέμισμα του πίνακα από το χρήστη με τη συνάρτηση `Fill_St_Arr(Student arr[],int size)`
- ✓ Εκτύπωση των στοιχείων του πίνακα με τη συνάρτηση `Print_St_Arr(Student arr[],int size)`
- ✓ Αξιολόγηση των μαθητών και εκτύπωση του αποτελέσματος με χρήση της συνάρτησης `Axiologhsh(Student arr[],int size)`



Συνάρτηση main()

```
int main()
{
    const int stud_num=4;
    Student s_array[stud_num];

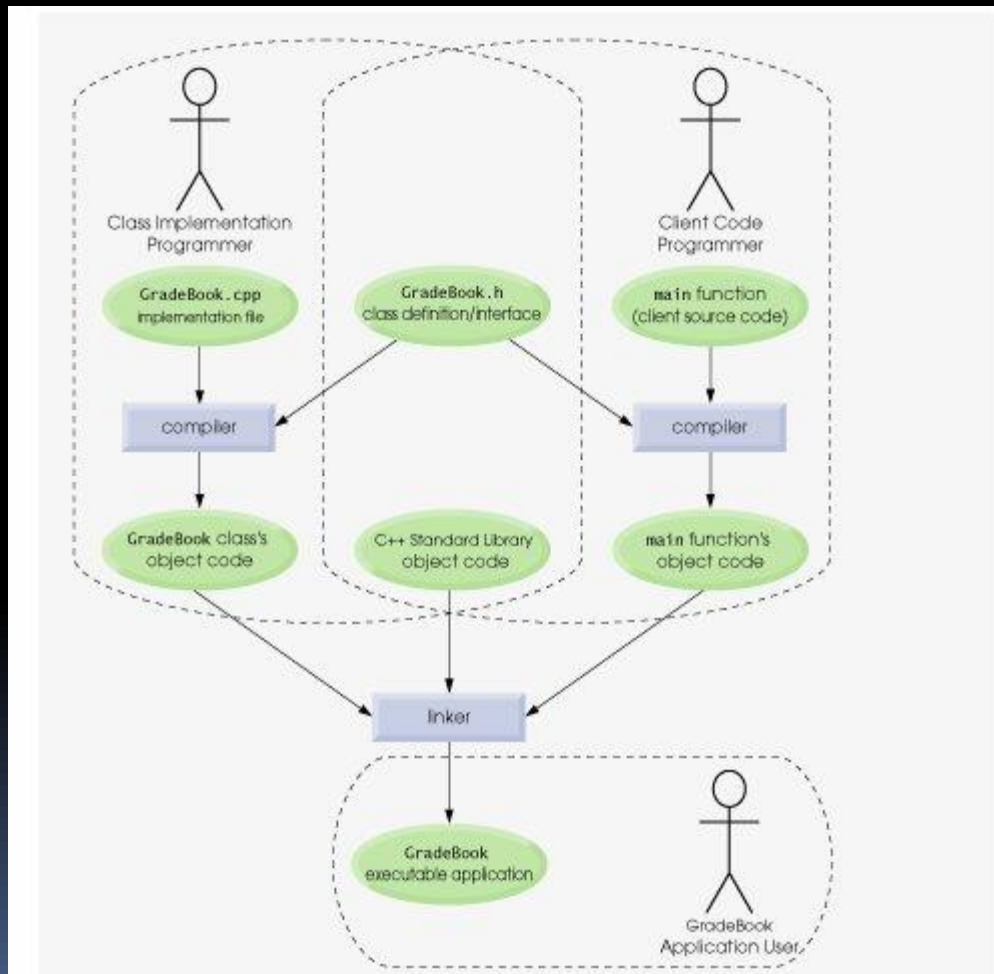
    Fill_St_Arr(s_array,stud_num);
    Print_St_Arr(s_array,stud_num);
    system("pause");

    Axiologhsh(s_array,stud_num);

    cout<<endl;
    system("pause");           // anamonh systhmatos
    return 0;                 // deixnei epityxh termatismo
}
```

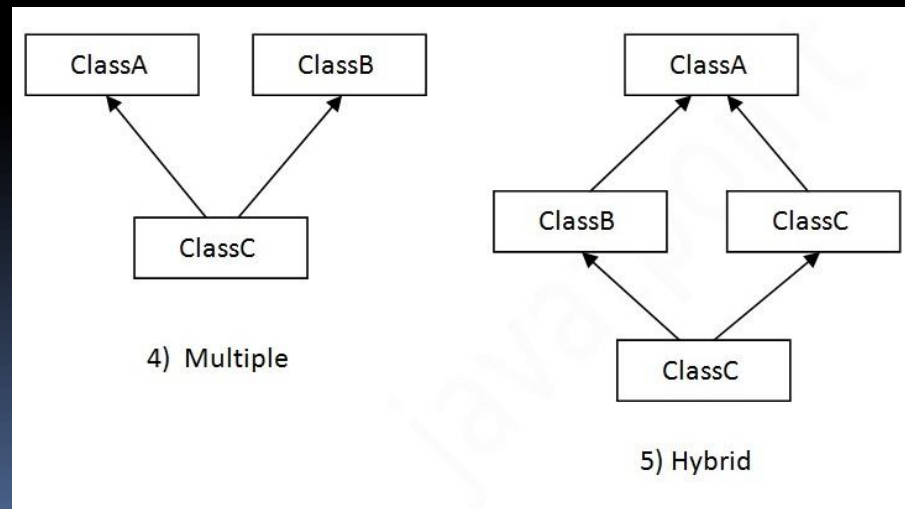
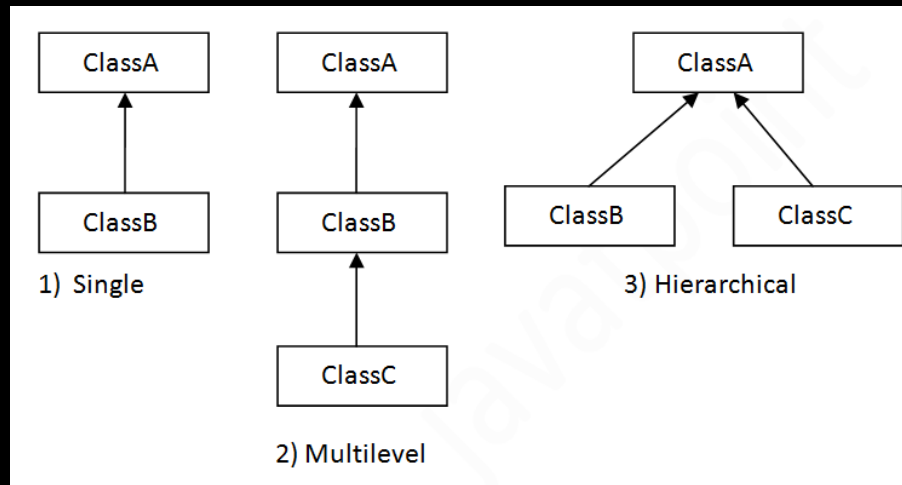
Ο πλήρης κώδικας της
άσκησης 2 είναι στο αρχείο
class2.cpp

Διαχωρισμός διασύνδεσης από υλοποίηση



Ο προγραμματιστής που κατασκεύασε την κλάση `GradeBook` παρέχει σε όποιον θέλει να την χρησιμοποιήσει το `.h` αρχείο και τον `object code` της κλάσης `GradeBook` (παράγεται από το `compile` του `.cpp` αρχείου). Δεν παρέχει τον πηγαίο κώδικα της κλάσης `GradeBook`

Κληρονομικότητα στη C++



Κληρονομικότητα στη C++

- Υποστηρίζει και τα 5 είδη κληρονομικότητας
- Συναρτήσεις friend και κατασκευαστές δεν κληρονομούνται
- Ορίζεται στη δήλωση της κλάσης-παιδί ως εξής:

Class **κλάση-παιδί**:public/protected/private **κλάση_γονέας**

```
class C:public A,public B {...}
```

Κληρονομικότητα public, protected, private

- Σπάνια χρησιμοποιούμε protected ή private
- **Public Inheritance:** ο,τι έχει δηλωθεί ως public στη κλάση-γονέα είναι public και στην κλάση-παιδί. Το ίδιο ισχύει και με τα protected μέλη. Τα private μέλη δεν είναι ποτέ προσβάσιμα από τη κλάση-παιδί.
- **Protected Inheritance:** τόσο τα public όσο και τα protected μέλη της κλάσης-γονέα γίνονται protected στην κλάση-παιδί.
- **Private Inheritance:** τα public όσο και τα protected μέλη της κλάσης-γονέα γίνονται private στην κλάση-παιδί.

```

#include<iostream.h>
#include<conio.h>

class student {
protected:
    int rno,m1,m2;
public:
    void get() {
        cout<<"Enter the Roll no:";
        cin>>rno;
        cout<<"Enter the two marks:";
        cin>>m1>>m2;
    }
};

class sports {
protected:
    int sm;                // sm = Sports mark
public:
    void getsm()          {
        cout<<"\nEnter the sports mark :";
        cin>>sm;
    }
};

class statement:public student,public sports {
    int tot,avg;
public:
    void display()        {
        tot=(m1+m2+sm);
        avg=tot/3;
        cout<<"\n\n\tRoll No: "<<rno<<"\n\tTotal: "<<tot;
        cout<<"\n\tAverage: "<<avg;
    }
};

void main() {
    clrscr();
    statement obj;
    obj.get();
    obj.getsm();
    obj.display();
    getch();
}

```

Output:

Enter the Roll no: 100

Enter two marks

90

80

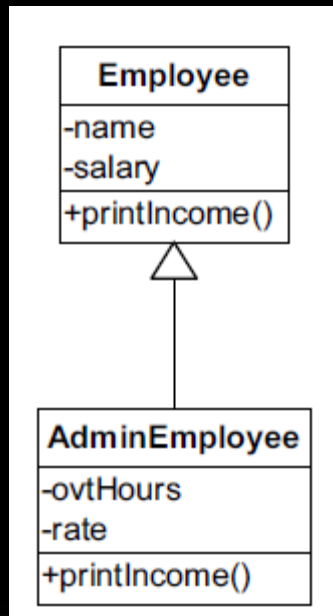
Enter the Sports Mark: 90

Roll No: 100

Total : 260

Average: 86.66

Κληρονομικότητα



```
#include <iostream>
#include <string>
using namespace std;

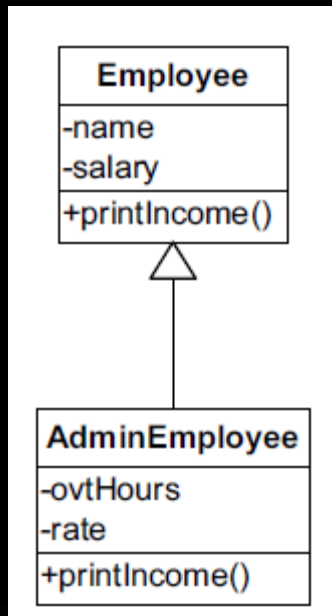
class Employee {
public:
    Employee(string text, double amount);
    void printIncome();

protected:
    string name;
    double salary;
};

Employee::Employee(string text, double amount)
{
    name = text;
    salary = amount;
}

void Employee::printIncome()
{
    cout << "Monthly Income for Employee: " << name
    << ": " << salary << endl;
}
```

Κληρονομικότητα



```
class AdminEmployee : public Employee {
public:
    AdminEmployee(string text, double amount,
                  int hours, double wage);
    void printIncome();

private:
    int ovtHours;
    double rate;
};

AdminEmployee::AdminEmployee(string text, double amount,
                             int hours, double wage) : Employee(text, amount)
{
    ovtHours = hours;
    rate = wage;
}

void AdminEmployee::printIncome()
{
    cout << "Monthly Income for AdminEmployee:" << name
         << ": " << salary + ovtHours * rate << endl;
}
```

```
int main()
{
    Employee E1("Fred", 1200);
    E1.printIncome();

    AdminEmployee E2("Bob", 1200, 15, 10);
    E2.printIncome();

    return 0;
}
```

➔ Το πρόγραμμα έχει την αναμενόμενη συμπεριφορά, δηλαδή η κλήση των μεθόδων `printIncome()` επί των αντίστοιχων αντικειμένων έχει ως αποτέλεσμα την εκτύπωση του αντίστοιχου μηνιαίου εισοδήματος για κάθε κατηγορία υπαλλήλων.

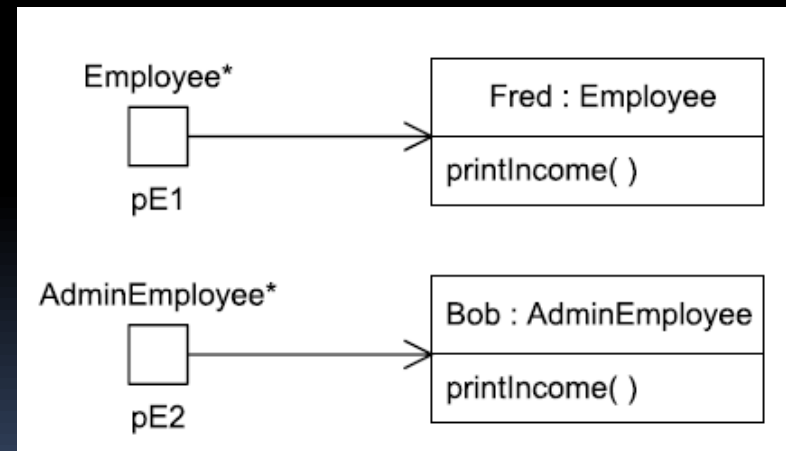
Κληρονομικότητα

```
int main()
{
    Employee* pE1 = new Employee("Fred", 1200);
    pE1->printIncome();

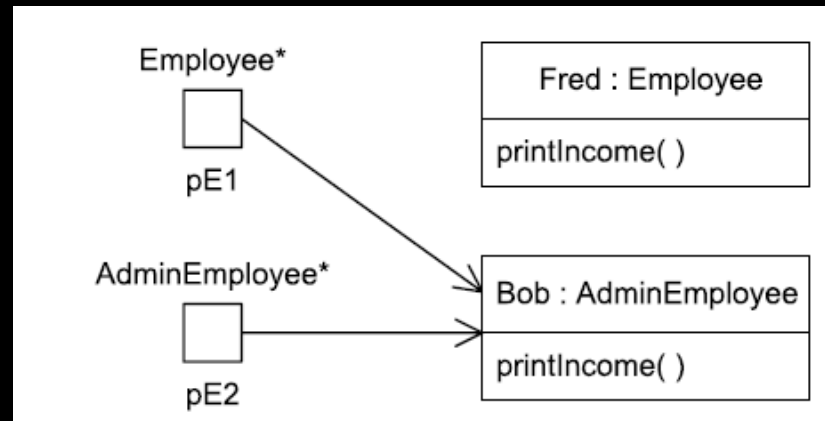
    AdminEmployee* pE2 = new AdminEmployee("Bob", 1200, 15, 10);
    pE2->printIncome();

    return 0;
}
```

Αν χρησιμοποιηθούν δείκτες για την κλήση των συναρτήσεων, η συμπεριφορά του προγράμματος δεν αλλάζει (παραμένει η αναμενόμενη)



Κληρονομικότητα



Επειδή κάθε αντικείμενο της κλάσης `AdminEmployee` αποτελεί ταυτοχρόνως και αντικείμενο της κλάσης `Employee`, στον δείκτη `pE1` που είναι δείκτης προς την βασική κλάση, είναι δυνατόν να ανατεθεί η διεύθυνση ενός αντικειμένου οποιασδήποτε υποκλάσης της `Employee`

Αρχής της υποκατάστασης: Σε δείκτες (ή αναφορές) προς μια βασική κλάση, είναι δυνατόν να ανατεθούν διευθύνσεις αντικειμένων οποιασδήποτε παράγωγης κλάσης.

Στατική διασύνδεση

```
int main()
{
    Employee* pE1 = new Employee("Fred", 1200);
    pE1->printIncome();

    AdminEmployee E2("Bob", 1200, 15, 10);

    pE1 = &E2; //ανάθεση διεύθυνσης αντικειμένου υποκλάσης
              //σε δείκτη προς τη βασική κλάση

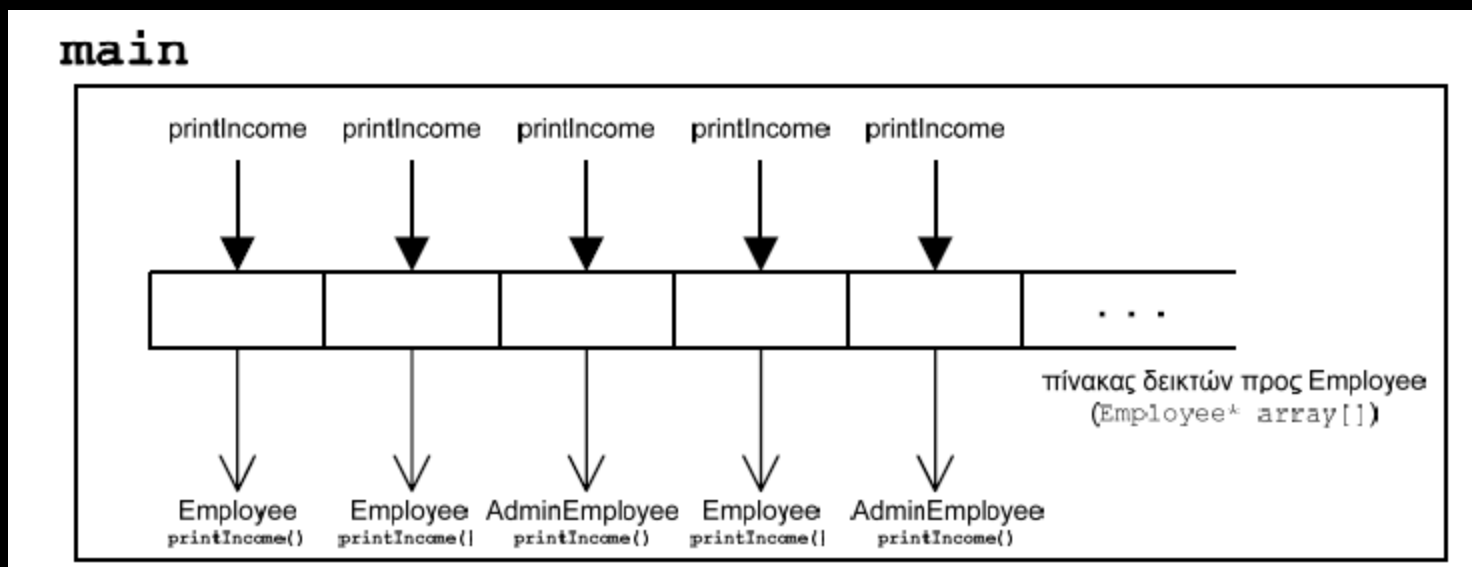
    pE1->printIncome(); //στατική διασύνδεση
                       //καλείται η printIncome() της Employee

    return 0;
}
```

Αν στο δείκτη pE1 προς τη βασική κλάση ανατεθεί η **διεύθυνση του αντικειμένου της παράγωγης κλάσης** η κλήση της μεθόδου printIncome() μέσω του δείκτη pE1 **θα εκτελέσει την printIncome() της βασικής κλάσης**

Αυτό είναι OK όταν η μέθοδος της βασικής κλάσης δεν επικαλύπτεται στην υποκλάση ή όταν ρητά απαιτείται η κλήση της μεθόδου της βασικής κλάσης

Πολυμορφισμός



Έστω όμως ότι:

Αν ένα στοιχείο του πίνακα "δείχνει" προς αντικείμενο τύπου `AdminEmployee`, παρόλο που ο πίνακας είναι δηλωμένος ως πίνακας δεικτών προς `Employee`, θα πρέπει να καλείται η μέθοδος της κλάσης `AdminEmployee`.

ΔΗΛΑΔΗ θέλουμε να αποστέλλεται **το ίδιο μήνυμα** (`printIncome`) σε κάθε στοιχείο του πίνακα, **αλλά η απόκριση να είναι διαφορετική** με βάση τον δείκτη που βρίσκεται αποθηκευμένος σε κάθε θέση

Πολυμορφισμός

Στη Java όλες οι μέθοδοι
είναι by default virtual
(δεν το δηλώνουμε)

Η δυνατότητα διασύνδεσης μιας μεθόδου με την αντίστοιχη κλάση κατά τη διάρκεια της εκτέλεσης καλείται δυναμική ή καθυστερημένη διασύνδεση (**dynamic or late binding**) και παρέχει τη δυνατότητα για πολυμορφική συμπεριφορά

Για να γίνει δυναμική διασύνδεση θα πρέπει η μέθοδος της βασικής κλάσης (που θέλουμε να έχει πολυμορφική συμπεριφορά) να δηλωθεί ως **υπερβατή (virtual)**.



virtual void printIncome()

```
int main()
{
    Employee* employees[5];

    employees[0] = new Employee("Fred", 1200);
    employees[1] = new AdminEmployee("Bob", 1200, 15, 10);
    employees[2] = new AdminEmployee("Nick", 800, 5, 15);
    employees[3] = new Employee("John", 1500);
    employees[4] = new AdminEmployee("Peter", 1000, 3, 20);

    for(int i=0; i<5; i++) {
        //πολυμορφική συμπεριφορά
        //(δυναμική διασύνδεση)
        //κατά τη μεταγλώττιση δεν είναι γνωστό
        //πού δείχνει ο δείκτης employees[i]
        //ώστε να διασυνδεθεί με την αντίστοιχη
        //μέθοδο κλάσης
        employees[i]->printIncome();
    }

    return 0;
}
```

Υπερφόρτωση, keywords CONST, STATIC, FRIEND

ΟΝΤΟΚΕΝΤΡΙΚΟΣ ΠΡΟΓΡ/ΣΜΟΣ C++

Μ. Ρήγκου (rigou@ceid.upatras.gr)

Τι θα συζητήσουμε σήμερα

- Υπερφόρτωση
 - Συναρτήσεις
 - Τελεστών
- CONST αντικείμενα, μεταβλητές και συναρτήσεις
- STATIC μέλη κλάσεων
- (Παράδειγμα με) friend συνάρτηση και διαχωρισμός της διεπαφής από την υλοποίηση

Υπερφόρτωση συναρτήσεων

- Η C++ μας δίνει την δυνατότητα να ορίσουμε **πολλές συναρτήσεις με το ίδιο όνομα** αρκεί να έχουν διαφορετικά σύνολα παραμέτρων (διαφορετικούς τύπους, διαφορετική σειρά, διαφορετικό πλήθος).
- Για να δούμε γιατί είναι χρήσιμη η υπερφόρτωση συναρτήσεων θυμηθείτε τις συναρτήσεις της C:
 - int **abs**(int j)
 - float **fabs**(double d)
 - long **labs**(long l)
- Παρόλο που οι συναρτήσεις κάνουν ουσιαστικά την ίδια δουλειά, είμαστε υποχρεωμένοι να διατηρούμε 3 ξεχωριστά ονόματα.

Υπερφόρτωση συναρτήσεων

- Στη C++ μπορούμε να διατηρήσουμε ένα όνομα

```
#include <iostream>
using namespace std;

int abs(int i) {
    cout << "using_integer_abs()" << endl;
    return i < 0 ? -i : i ;
}

double abs(double d) {
    cout << "using_double_abs()" << endl;
    return d < 0.0 ? -d : d ;
}

long abs(long l) {
    cout << "using_long_abs()" << endl;
    return l < 0 ? -l : l ;
}

int main() {
    cout << abs(-10) << endl;
    cout << abs(-11.0) << endl;
    cout << abs(-9L) << endl;
}
```

Για να κάνουμε υπερφόρτωση παραμέτρων πρέπει ο **τύπος και/ή ο αριθμός των παραμέτρων** κάθε υπερφορτωμένης συνάρτησης να διαφέρουν. **Δεν αρκεί να διαφέρουν μόνο στον τύπο επιστροφής**

```
using integer abs()
10
using double abs()
11
using long abs()
9
```

Υπερφόρτωση συναρτήσεων


```
#include <iostream>

// volume of a cube
int volume(int s)
{
    return s*s*s;
}

// volume of a cylinder
double volume(double r, int h)
{
    return 3.14*r*r*static_cast<double>(h);
}

// volume of a cuboid
long volume(long l, int b, int h)
{
    return l*b*h;
}


int main()
{
    std::cout << volume(10);
    std::cout << volume(2.5, 8);
    std::cout << volume(100, 75, 15);
}
```



```
1000
157
112500
```

```
#include <iostream.h>
class Overclass
{
public:
    int x;
    int y;
    Overclass() { x = y = 0; }
    Overclass(int a) { x = y = a; }
    Overclass(int a, int b) { x = a; y = b; }
};

int main()
{
    Overclass A;
    Overclass A1(4);
    Overclass A2(8, 12);
    cout << "Overclass A's x,y value:: " <<
    A.x << " , " << A.y << "\n";
    cout << "Overclass A1's x,y value:: " <<
    A1.x << " , " << A1.y << "\n";
    cout << "Overclass A2's x,y value:: " <<
    A2.x << " , " << A2.y << "\n";
    return 0;
}
```



```
Overclass A's x,y value:: 0 , 0
Overclass A1's x,y value:: 4 ,4
Overclass A2's x,y value:: 8 , 12
```

Υπερφόρτωση Τελεστών

- Η C++ μας επιτρέπει να υπερφορτώσουμε τελεστές π.χ +, -, κ.τ.λ ώστε να συμπεριφέρονται διαφορετικά ανάλογα με το αντικείμενο στο οποίο εφαρμόζονται
- Για παράδειγμα ο τελεστής << όταν χρησιμοποιείται με το cout είναι εξ' ορισμού υπερφορτωμένος (λειτουργεί με όλους τους προκαθορισμένους τύπους, pointers και strings)
- Τελεστές στους οποίους μπορεί να γίνει υπερφόρτωση
+ - * / % ^ & | ~ != < > += -= *= /= %= ^= &=
|= << >> <<= >>= == != <= >= && || ++ -- -> ->*
[] () ,
- Στους τελεστές + - * & μπορούμε να κάνουμε υπερφόρτωση τόσο στη μοναδιαία (ένας τελεσταίος) όσο και στην δυαδική τους μορφή (δύο τελεσταίοι)

Υπερφόρτωση Τελεστών

- Τελεστές στους οποίους δεν μπορεί να γίνει υπερφόρτωση

. * :: ?:

- Δεν μπορούμε να δημιουργήσουμε νέους τελεστές
- Μία συνάρτηση υπερφόρτωσης πρέπει να έχει τουλάχιστο ένα όρισμα (implicit ή explicit) τύπου κλάσης.
- Η προτεραιότητα και η προσεταιριστικότητα των υπερφορτωμένων τελεστών παραμένει η ίδια με αυτή που διαθέτουν στην προκαθορισμένη τους μορφή

Υπερφόρτωση Τελεστών

- Χρειάζεται ιδιαίτερη προσοχή στην υπερφόρτωση των τελεστών ++ και -- και αυτό γιατί έχουμε και προθεματική μορφή (++x) και επιθεματική μορφή (x++) αυτών των τελεστών.
- Αν θέλουμε να καλύψουμε και τις δύο περιπτώσεις πρέπει να ορίσουμε δύο συναρτήσεις υπερφόρτωσης

```
operator++();          //prefix inc
```

και

```
operator++(int);      //postfix
```

Στην περίπτωση της συνάρτησης υπερφόρτωσης για την επιθεματική εκδοχή το όρισμα `int` χρησιμοποιείται μόνο για να δείξει ότι πρόκειται για επιθεματική μορφή. ΠΡΟΣΟΧΗ: Δεν είναι ένας `int` που πρέπει να καθορίσουμε κατά την κλήση αλλά ένα `dummy argument`

Προβληματική υπερφόρτωση των ++, --

```
#include<iostream>
using namespace std;

//Increment and decrement overloading
class Inc {
private:
    int count ;
public:
    Inc() {
        //Default constructor
        count = 0 ;
    }

    Inc(int C) {
        // Constructor with Argument
        count = C ;
    }

    Inc operator ++ () {
        // Operator Function Definition
        return Inc(++count);
    }

    Inc operator -- () {
        // Operator Function Definition
        return Inc(--count);
    }

    void display(void) {
        cout << count << endl ;
    }
};
```

```
void main(void) {
    Inc a, b(4), c, d, e(1), f(4);

    cout << "Before using the operator ++()\n";
    cout << "a = ";
    a.display();
    cout << "b = ";
    b.display();

    ++a;
    b++;

    cout << "After using the operator ++()\n";
    cout << "a = ";
    a.display();
    cout << "b = ";
    b.display();

    c = ++a;
    d = b++;

    cout << "Result prefix (on a) and postfix (on b)\n";
    cout << "c = ";
    c.display();
    cout << "d = ";
    d.display();

    cout << "Before using the operator --()\n";
    cout << "e = ";
    e.display();
    cout << "f = ";
    f.display();

    --e;
    f--;

    cout << "After using the operator --()\n";
    cout << "e = ";
    e.display();
    cout << "f = ";
    f.display();
}
```

Προθεματική & επιθεματική υπερφόρτωση των ++, --

```
#include<iostream>
using namespace std;

//Increment and decrement overloading
class Inc {
private:
    int count ;
public:
    Inc() {
        //Default constructor
        count = 0 ;
    }

    Inc(int C) {
        // Constructor with Argument
        count = C ;
    }

    Inc operator ++ () {
        // Operator Function Definition
        // for prefix
        return Inc(++count);
    }

    Inc operator ++ (int) {
        // Operator Function Definition
        // with dummy argument for postfix
        return Inc(count++);
    }

    Inc operator -- () {
        // Operator Function Definition
        // for prefix
        return Inc(--count);
    }

    Inc operator -- (int) {
        // Operator Function Definition
        // with dummy argument for postfix
        return Inc(count--);
    }

    void display(void) {
        cout << count << endl ;
    }
};
```

CONST αντικείμενα και συναρτήσεις

- Principle of least privilege
- CONST: δηλώνει ότι το αντικείμενο δε μπορεί να μεταβληθεί
 - Οποιαδήποτε απόπειρα μεταβολής του αντικειμένου που έχει δηλωθεί const προκαλεί compile error
 - Αυξάνει την απόδοση του κώδικα
- Η C++ δεν επιτρέπει σε member functions να έχουν πρόσβαση σε μεταβλητές const παρά μόνο αν δηλωθούν και οι functions const
 - Αυτό ισχύει και για getters const μεταβλητών (που δεν αλλάζουν τιμές)
- Μια const member function μπορεί να υπερφορτωθεί με μια non-const εκδοχή της. Ανάλογα με το είδος του αντικειμένου στο οποίο καλείται (const ή όχι) επιλέγεται η αντίστοιχη έκδοση.
- Δε δηλώνουμε const constructors ή destructors (συντακτικό λάθος)

Σταθερά Αντικείμενα και Σταθερές Συναρτήσεις Μέλη (`const`)

Άσκηση (`class3.cpp`)

- Στην υλοποίηση της κλάσης `part` δώστε την δυνατότητα στο πρόγραμμα να εκτυπώνει τις τιμές των δεδομένων σταθερών (`constant`) αντικείμενων τύπου `part` κάνοντας τις απαραίτητες αλλαγές.
- Στη συνέχεια δημιουργήστε ένα σταθερό αντικείμενο τύπου `part` με όνομα `"shifter"` το οποίο θα αναπαριστά ένα λεβιέ ταχυτήτων.
 - ✓ Εκτυπώστε τις τιμές δεδομένων του αντικειμένου
- Στη συνέχεια δημιουργήστε ένα σταθερό αντικείμενο τύπου `part` με όνομα `"mirror"` το οποίο θα αναπαριστά έναν καθρέπτη δίνοντας τιμές στα δεδομένα του (`modelName=8030`, `partNumber=8020`, `cost=100.0`).
 - ✓ Εκτυπώστε τις τιμές δεδομένων του αντικειμένου

Σημείωση: Στα αρχεία `.cpp` και `.h`, όπου υπάρχει η σήμανση `/**/` σημαίνει ότι εκεί έγιναν οι αλλαγές στον κώδικα για την υλοποίηση κάθε άσκησης σε σχέση με την προηγούμενη.

Μέλη δεδομένων `const` - Αρχικοποίηση

Άσκηση (`class4.cpp`)

- Συνεχίστε την υλοποίηση της κλάσης `part`.
- Να δημιουργηθεί ένα νέο σταθερό (`constant`) μέλος δεδομένων της κλάσης `part` το οποίο να εκφράζει την εταιρία από την οποία αγοράστηκε κάθε ανταλλακτικό.
- Να δημιουργηθεί ένα αντικείμενο τύπου `part` με όνομα "gear" το οποίο θα αναπαριστά μια ταχύτητα δίνοντας τιμές στα δεδομένα του (`modelNumber=2020, partNumber=2025, cost=300.0, compqny=2`).
- Εκτύπωση τιμών δεδομένων αντικειμένου
- Ας υποθέσουμε ότι η τιμή του ανταλλακτικού `gear` πρέπει να αλλάξει λόγω νέας τιμολόγησης του προϊόντος σε `350.0` και λόγω νέας οργάνωσης της αποθήκης οι κωδικοί του προϊόντος να πάρουν νέες τιμές (`modelNumber=3030, partNumber=3025`). Να γίνουν οι αλλαγές και να τυπωθούν τα νέα χαρακτηριστικά του ανταλλακτικού `gear`.

STATIC Μέλη δεδομένων

- Ακριβώς η ίδια έννοια όπως και στη Java
- Μια static ιδιότητα αποθηκεύει δεδομένα που είναι προσβάσιμα από όλα τα αντικείμενα της κλάσης (και υπάρχουν σε επίπεδο κλάσης)

```
class Student
{
    public:
        Student()
        {
            noOfStudents++; // reference from inside the class
            // ...other stuff...
        }

        static int noOfStudents;
        // ...other stuff like before...
};

void fn(Student& s1, Student& s2)
{
    // reference public static
    cout << "No of students "
         << s1.noOfStudents // reference from outside
         << endl;           // of the class
}
```

```
// ...class defined the same as before...
void fn(Student& s1, Student& s2)
{
    // the following produce identical results
    cout << " Number of students "
         << s1.noOfStudents
         << endl;
    cout << " Number of students "
         << s2.noOfStudents
         << endl;
}
```

```
// ...class defined the same as before...
void fn(Student& s1, Student& s2)
{
    // the following produce identical results
    cout << "Number of students "
         << Student::noOfStudents
         << endl;
}
```

- Μπορούμε να δηλώσουμε και static member functions

Static μέλη συναρτήσεις

- Είναι κατ' αντιστοιχία ανεξάρτητες από κάποιο συγκεκριμένο αντικείμενο και μπορούν να καλούνται επί της κλάσης
- Έχουν πρόσβαση μόνο σε static μεταβλητές, άλλες static συναρτήσεις ή σε συναρτήσεις εκτός κλάσης

```
class user
{
    private:
        int id;
        static int next_id;
    public:
        static int next_user_id() { next_id++; return next_id; }
        /* More stuff for the class user */
        user() { id = user::next_id++; //or, id = user.next_user_id(); }
};
int user::next_id = 0;
```

Στατικά (*static*) μέλη κλάσεων

Άσκηση (`class5.cpp`)

- Συνεχίστε την υλοποίηση της κλάσης `part`.
- Να δημιουργηθεί ένα στατικό μέλος της κλάσης `part` το οποίο να είναι ένας ακέραιος αριθμός ο οποίος θα εκφράζει πόσα ανταλλακτικά έχουν καταχωρηθεί μέχρι στιγμής στην αποθήκη (δηλ. πόσα αντικείμενα τύπου `part` έχουν δημιουργηθεί μέσα στο πρόγραμμα). Να ονομαστεί `counter`.
- Να τυπωθεί η τιμή του `counter` στη `main`.
- Να δημιουργηθούν 3 αντικείμενα τύπου `part` και να εκτυπωθούν οι τιμές των δεδομένων τους.
- Να τυπωθεί η νέα τιμή του `counter`.

Header files

- Δεν επιτρέπεται να έχουμε πολλαπλούς ορισμούς της ίδιας συνάρτησης ή της ίδιας μεταβλητής

```
int main()
{
    int x; // δήλωση της μεταβλητής x
    int x; // compile error: duplicate definition
    return 0;
}
```

- Το ίδιο ισχύει και για τη συμπερίληψη header files. Γι' αυτό χρησιμοποιούμε header guards:

```
#ifndef SOME_UNIQUE_NAME_HERE
#define SOME_UNIQUE_NAME_HERE

// your declarations and definitions here

#endif
```

Αν το συγκεκριμένο αρχείο έχει ήδη συμπεριληφθεί τότε δεν θα εκτελεστεί το #include

Φιλική Συνάρτηση (Friend Function)

Άσκηση

(class6.cpp)

- Συνεχίστε την υλοποίηση της κλάσης part
- Να δημιουργηθεί μια φιλική συνάρτηση στην κλάση part η οποία θα παίρνει σαν όρισμα ένα αντικείμενο τύπου part, θα μηδενίζει τις τιμές των ιδιωτικών δεδομένων modelNumber, partNumber και cost του αντικειμένου αυτού και θα τυπώνει στην οθόνη τα νέα χαρακτηριστικά του αντικειμένου. Η συνάρτηση αυτή θα έχει το όνομα zero.
- Δημιουργία ενός αντικειμένου τύπου part με όνομα "wheel25" με αρχικοποίηση των τιμών των δεδομένων του με τις εξής τιμές :
modelNumber=4036 ,partNumber=4026, cost=350.0,company=3.
- Εκτύπωση τιμών δεδομένων αντικειμένου με τη show_part
- Κλήση της zero για το αντικείμενο.
- Εκτύπωση τιμών δεδομένων (μόνο των modelNumber, partNumber και cost) του αντικειμένου με τη χρήση των get_modelNumber, get_partNumber, get_cost



Πέρασμα μεταβλητών, Templates, Συσχετίσεις μεταξύ κλάσεων

ΟΝΤΟΚΕΝΤΡΙΚΟΣ ΠΡΟΓΡ/ΣΜΟΣ C++

Μ. Ρήγκου (rigou@ceid.upatras.gr)

Τι θα συζητήσουμε σήμερα

- Πέρασμα με τιμή και με αναφορά
- Template functions και classes
- STL
 - Vector
 - List
- Σχέσεις μεταξύ κλάσεων
 - Κληρονομικότητα
 - Απλή συσχέτιση
 - Συνάθροιση/συναρμολόγηση
 - Σύνθεση

Πέρασμα ορίσματος με τιμή

- Εξ' ορισμού μηχανισμός στη C++
 - Αν μια μεταβλητή αλλάξει τιμή μέσα σε μια συνάρτηση η τιμή της δεν αλλάζει στην συνάρτηση που την κάλεσε

```
void fn(int nArg)
{
    nArg = 10;
    // value of nArg at this point is 10
}

void parent(void)
{
    int n1 = 0;
    fn(n1);
    // value of n1 at this point is still 0
}
```

Δείκτης σαν όρισμα συνάρτησης

<i>Operator</i>	<i>Meaning</i>
<code>&(unary)</code>	(In an expression) the address of
<code>&(unary)</code>	(In a declaration) reference to
<code>*(unary)</code>	(In an expression) the thing pointed at by
<code>*(unary)</code>	(In a declaration) pointer to

```
int *x;  
int y;
```

```
&y // διεύθυνση του y  
*x // τιμή που δείχνεται από το x  
// (de-referencing: από-αναφοροποίηση)
```

```
int myVar;  
int *myPtr = &myVar;
```

```
void fn(int* pnArg)  
{  
    *pnArg = 10;  
}  
  
void parent(void)  
{  
    int n = 0;  
  
    fn(&n); // this passes the address  
           // now the value of n is 10  
}
```

← Πέρασμα με αναφορά

Πέρασμα ορίσματος με αναφορά

Η C++ μας επιτρέπει να υλοποιήσουμε πέρασμα με αναφορά χωρίς δείκτες

```
int n1;           // declare an int variable
int& nRef = n1;  // declare a second reference to n1

nRef = 1;        // now accessing the reference
                // has the same effect as accessing n1;
                // n1 is now equal to 1
```

```
void fn(int& rnArg) // declare reference argument
{
    rnArg = 10;     // change the value of the variable...
}                 //...that rnArg refers to

void parent(void)
{
    int n1 = 0;
    fn(n1);        // pass a reference to n1
                  // here the value of n1 is 10
}
```

- Δε μπορούμε να υπερφορτώσουμε την ίδια συνάρτηση με πέρασμα με τιμή και με αναφορά
 - `fn(int)` και `fn(int&)`

Ας περάσουμε στα αντικείμενα: Πίνακας αντικειμένων

```
// ArrayOfStudents - define an array of student objects
//                    and access an element in it. This
//                    program doesn't do anything
#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

class Student
{
public:
    int semesterHours;
    double gpa;
    double addCourse(int hours, double grade);
};
```

```
void someFn()
{
    // declare an array of 10 students
    Student s[10];

    // assign the 5th student a gpa of 4.0 (lucky guy)
    s[4].gpa = 4.0;
    s[4].semesterHours = 32;

    // add another course to the 5th student;
    // this time he failed - serves him right
    s[4].addCourse(3, 0.0);
}
```

Δείκτες προς αντικείμενα

```
// ObjPtr - define and use a pointer to a Student object
#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

class Student
{
public:
    int semesterHours;
    double gpa;
    double addCourse(int hours, double grade);
};

int main(int argc, char* pArgs[])
{
    // create a Student object
    Student s;
    s.gpa = 3.0;

    // now create a pointer pS to a Student object
    Student* pS;

    // make pS point to our Student object
    pS = &s;

    // now output the gpa of the object, once thru
    // the variable name and a second time thru pS
    cout << "s.gpa = " << s.gpa << "\n"
         << "pS->gpa = " << pS->gpa << endl;

    // wait until user is ready before terminating program
    // to allow the user to see the program results
    cout << "Press Enter to continue..." << endl;
    cin.ignore(10, '\n');
    cin.get();
    return 0;
}
```


Λειτουργούν όπως και οι δείκτες προς απλούς τύπους δεδομένων (int, char, κτλ.)

Από-αναφοροποίηση δείκτη σε αντικείμενα

```
int main(int argc, char* pArgs[])
{
    Student s;
    Student* pS = &s; // create a pointer to s

    // access the gpa member of the obj pointed at by pS
    // (this doesn't work)
    *pS.gpa = 3.5;

    return 0;
}
```

- Δε λειτουργεί σωστά επειδή ο τελεστής (.) έχει προτεραιότητα σε σχέση με τον (*), δηλαδή πρακτικά:
 - *(pS.gpa)
 - Άρα για να λειτουργήσει ο κώδικας  (*pS).gpa
- Είναι δύσχρηστο να χρησιμοποιούμε (*pS).gpa ενώ είναι πολύ πιο διαισθητικό το
 - pS->gpa

Κλήση συνάρτησης με τιμή αντικειμένου

```
// PassObjVal - attempts to change the value of an object
//                in a function fail when the object is
//                passed by value
#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

class Student
{
public:
    int semesterHours;
    double gpa;
};

void someFn(Student copyS)
{
    copyS.semesterHours = 10;
    copyS.gpa           = 3.0;
    cout << "The value of copyS.gpa = " << copyS.gpa << endl;
}

int main(int argc, char* pArgs[])
{
    Student s;
    s.gpa = 0.0;

    // display the value of s.gpa before calling someFn()
    cout << "The value of s.gpa = " << s.gpa << endl;

    // pass the address of the existing object
    cout << "Calling someFn(Student)" << endl;
    someFn(s);
    cout << "Returned from someFn(Student)" << endl;

    // the value of s.gpa remains 0
    cout << "The value of s.gpa = " << s.gpa << endl;

    // wait until user is ready before terminating program
    // to allow the user to see the program results
    cout << "Press Enter to continue..." << endl;
    cin.ignore(10, '\n');
    cin.get();
    return 0;
}
```

The value of s.gpa = 0
Calling someFn(Student)
The value of copyS.gpa = 3
Returned from someFn(Student)
The value of s.gpa = 0
Press Enter to continue...



Κλήση συνάρτησης με δείκτη σε αντικείμενο

```
The value of s.gpa = 0
Calling someFn(Student*)
The value of pS->gpa = 3
Returned from someFn(Student*)
The value of s.gpa = 3
Press Enter to continue...
```

```
// PassObjPtr - change the contents of an object in
// a function by passing a pointer
#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

class Student
{
public:
    int semesterHours;
    double gpa;
};

void someFn(Student* pS)
{
    pS->semesterHours = 10;
    pS->gpa = 3.0;
    cout << "The value of pS->gpa = " << pS->gpa << endl;
}

int main(int nNumberOfArgs, char* pszArgs[])
{
    Student s;
    s.gpa = 0.0;

    // display the value of s.gpa before calling someFn()
    cout << "The value of s.gpa = " << s.gpa << endl;

    // pass the address of the existing object
    cout << "Calling someFn(Student*)" << endl;
    someFn(&s);
    cout << "Returned from someFn(Student*)" << endl;

    // the value of s.gpa is now 3.0
    cout << "The value of s.gpa = " << s.gpa << endl;

    // wait until user is ready before terminating program
    // to allow the user to see the program results
    cout << "Press Enter to continue..." << endl;
    cin.ignore(10, '\n');
    cin.get();
    return 0;
}
```

Κλήση συνάρτησης με τον τελεστή αναφοράς

Ίδια έξοδος όπως
στην κλήση με
δείκτη

```
// PassObjRef - change the contents of an object in
// a function by using a reference
#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

class Student
{
public:
    int semesterHours;
    double gpa;
};

// same as before, but this time using references
void someFn(Student& refS)
{
    refS.semesterHours = 10;
    refS.gpa = 3.0;
    cout << "The value of copyS.gpa = " << refS.gpa << endl;
}

int main(int nNumberOfArgs, char* pszArgs[])
{
    Student s;
    s.gpa = 0.0;

    // display the value of s.gpa before calling someFn()
    cout << "The value of s.gpa = " << s.gpa << endl;

    // pass the address of the existing object
    cout << "Calling someFn(Student*)" << endl;
    someFn(s);
    cout << "Returned from someFn(Student&)" << endl;

    // the value of s.gpa is now 3.0
    cout << "The value of s.gpa = " << s.gpa << endl;

    // wait until user is ready before terminating program
    // to allow the user to see the program results
    cout << "Press Enter to continue..." << endl;
    cin.ignore(10, '\n');
    cin.get();
    return 0;
}
```

Γιατί χρειάζεται να χρησιμοποιούμε πέρασμα με αναφορά;

- Η C++ υλοποιεί πέρασμα με τιμή όταν δίνουμε σαν όρισμα ένα αντικείμενο
 - Άρα δε μπορούμε να αλλάξουμε το αντικείμενο-όρισμα, αφού η καλούσα συνάρτηση χρησιμοποιεί ένα αντίγραφό του
- Ένα αντικείμενο μπορεί να είναι μεγάλο σε μέγεθος και η αντιγραφή του να επιβαρύνει πολύ τους πόρους του συστήματος
 - ιδιαίτερα αν η συνάρτηση που το έχει όρισμα καλεί κι άλλες συναρτήσεις (και άρα το αντικείμενο ξανα-αντιγράφεται)
- Η αντιγραφή ενός δείκτη ή μιας αναφοράς είναι πολύ γρήγορη
 - ένας δείκτης είναι 4 bytes, ανεξάρτητα από το αντικείμενο στο οποίο δείχνει



TEMPLATES

Template functions και template classes



Γιατί χρειάζονται τα templates;

- Υπάρχουν συναρτήσεις που μπορούν να εφαρμοστούν σε διάφορους τύπους δεδομένων

```
int maximum(int n1, int n2); // return max of two integers
unsigned maximum (unsigned u1, unsigned u2);
double maximum (double d1, double d2);
char maximum (char c1, char c2);
```

Μια λύση:

```
double maximum(double d1, double d2)
{
    return (d1 > d2) ? d1:d2;
}
int maximum(int n1, int n2)
{
    return (n1 > n2) ? n1:n2;
}
char maximum(char c1, char c2)
{
    return (c1 > c2) ? c1:c2;
}

// ...repeat for all other numeric types...
```

Υπερφόρτωση:

Γράφουμε τον ίδιο κώδικα τόσες φορές όσοι οι τύποι ορισμάτων που θέλουμε να λειτουργούν με τη συνάρτηση

Μια καλύτερη λύση: Templates

- Τα templates μας επιτρέπουν να γράψουμε συναρτήσεις που χρησιμοποιούν έναν ή περισσότερους **type holders**
- Οι type holders μετατρέπονται σε **true types** κατά το compiling


```
// MaxTemplate - create a template max() function
//                that returns the greater of two types
#include <stdio>
#include <stdlib>
#include <iostream>

using namespace std;

template <class T> T maximum(T t1, T t2)
{
    return (t1 > t2) ? t1 : t2;
}

int main(int argc, char* pArgs[])
{
    // find the maximum of two int's;
    // here C++ creates maximum(int, int)
    cout << "maximum(-1, 2) = " << maximum(-1, 2) << endl;
    // repeat for two doubles;
    // in this case, we have to provide T explicitly since
    // the types of the arguments are different
    cout << "maximum(1, 2.5) = " << maximum<double>(1, 2.5)
        << endl;

    cout << "Press Enter to continue..." << endl;
    cin.ignore(10, '\n');
    cin.get();
    return 0;
}
```



```
maximum(-1, 2) = 2
maximum(1, 2.5) = 2.5
Press Enter to continue...
```

Templates κλάσεων

templatevector.h

- Ακολουθεί τον ορισμό μιας τυπικής κλάσης αλλά χρησιμοποιεί **placeholder** για κάποιες άγνωστες βοηθητικές κλάσεις

```
// TemplateVector - a simple templated vector class
template <class T>
class TemplateVector
{
public:
    TemplateVector(int nArraySize)
    {
        // store off the number of elements
        nSize = nArraySize;
        array = new T[nArraySize];
        reset();
    }
    int size() { return nWriteIndex; }
    void reset() { nWriteIndex = 0; nReadIndex = 0; }
    void add(const T& object)
    {
        if (nWriteIndex < nSize)
        {
            array[nWriteIndex++] = object;
        }
    }
    T& get()
    {
        return array[nReadIndex++];
    }

protected:
    int nSize;
    int nWriteIndex;
    int nReadIndex;
    T* array;
};
```

Class templates: χρήση

```
// TemplateVector - implement a vector that uses a
// template type
#include <cstdlib>
#include <cstdio>
#include <iostream>
#include "templatevector.h"
using namespace std;

// intFn() - manipulate a collection of integers
void intFn()
{
    // create a vector of integers
    TemplateVector<int> integers(10);

    // add values to the vector
    cout << "Enter integer values to add to a vector\n"
         << "(Enter a negative number to terminate):"
         << endl;
    for(;;)
    {
        int n;
        cin >> n;

        if (n < 0) { break; }
        integers.add(n);
    }

    cout << "\nHere are the numbers you entered:" << endl;
    for(int i = 0; i < integers.size(); i++)
    {
        cout << i << ":" << integers.get(i) << endl;
    }
}

// Names - create and manipulate a vector of names
class Name
{
public:
    Name(string s) : name(s) {}
    const string& display() { return name; }
protected:
    string name;
};
```

```
void nameFn()
{
    // create a vector of Name objects
    TemplateVector<Name> names(20);

    // add values to the vector
    cout << "Enter names to add to a second vector\n"
         << "(Enter an 'x' to quit):" << endl;
    for(;;)
    {
        string s;
        cin >> s;
        if (s == "x" || s == "X") { break; }
        names.add(Name(s));
    }

    cout << "\nHere are the names you entered" << endl;
    for(int i = 0; i < names.size(); i++)
    {
        Name& name = names.get(i);
        cout << i << ":" << name.display() << endl;
    }
}

int main(int argc, char* pArgs[])
{
    intFn();
    nameFn();

    cout << "Press Enter to continue..." << endl;
    cin.ignore(10, '\n');
    cin.get();
    return 0;
}
```


Διευκρινήσεις για τα templates

- Δεν παράγεται κώδικας για ένα template
 - Ο κώδικας παράγεται αφού δημιουργήσουμε ένα στιγμιότυπο του template δηλ. κατασκευάσουμε μια συγκεκριμένη κλάση ή συνάρτηση από αυτό
 - Για το λόγο αυτό σχεδόν πάντα ο ορισμός ενός class template μπαίνει σε .h αρχείο με τον κώδικα όλων των member functions
- Ένα class template δεν καταναλώνει μνήμη αν δεν παραχθούν συγκεκριμένες κλάσεις
- Ένα template δε γίνεται compiled μέχρι να μετατραπεί σε κλάση (μέχρι τότε δε μπορούμε να εντοπίσουμε τυχόν λάθη)



STL

vector, list



STL: Standard Templates Library

- Προσφέρει ένα σύνολο από αφηρημένους τύπους δεδομένων, συναρτήσεις και αλγόριθμους (searching, sorting, κ.ά.) σχεδιασμένους να λειτουργούν με user-specified τύπους δεδομένων
- Κάθε αφηρημένος τύπος δεδομένων παρέχει υλοποιημένες χρήσιμες συναρτήσεις, συμπεριλαμβανομένων συναρτήσεων υπερφόρτωσης τελεστών
- Ιδέα generic programming – η υλοποίηση των αλγορίθμων ή των δομών δεδομένων είναι ανεξάρτητη από τον τύπο των δεδομένων που διαχειρίζονται
- Παρέχει επιπλέον σημαντικά πλεονεκτήματα:
 - Διαχείριση μνήμης (no memory leaks)
 - Ασφάλεια (no buffer overflow)

STL: Standard Templates Library

Component	Description
Containers	Containers are used to manage collections of objects of a certain kind. There are several different types of containers like deque, list, vector, map etc.
Algorithms	Algorithms act on containers. They provide the means by which you will perform initialization, sorting, searching, and transforming of the contents of containers.
Iterators	Iterators are used to step through the elements of collections of objects. These collections may be containers or subsets of containers.

Η κλάση `vector`

- Μια από τις βασικές κλάσεις της STL
- Ένα `vector` είναι πρακτικά ένας `resizable` πίνακας
- Η κλάση μας παρέχει υλοποιημένες συναρτήσεις για **random access** μέσω του τελεστή `[]`
- Η προσθήκη ενός στοιχείου σε τυχαία θέση (και όχι μετά το τελευταίο) προκαλεί κάποιο `overhead` (μετακίνηση στοιχείων)
- Οι απαιτήσεις σε αποθηκευτικό χώρο είναι αντίστοιχες με αυτές του απλού `array`
- Κάνουμε `include` το header file `vector` (**`#include <vector>`**)
- Η κλάση `vector` ανήκει στο `std namespace` (**`using namespace std;`**)

<code>unsigned int size();</code>	Returns the number of elements in a vector
<code>push_back(type element);</code>	Adds an element to the end of a vector
<code>bool empty();</code>	Returns true if the vector is empty
<code>void clear();</code>	Erases all elements of the vector
<code>type at(int n);</code>	Returns the element at index n, with bounds checking

`=` Assignment replaces a vector's contents with the contents of another
`==` An element by element comparison of two vectors
`[]` Random access to an element of a vector (usage is similar to that of the operator with arrays.) Keep in mind that it does not provide bounds checking.

```
#include <iostream>
#include <vector>

using namespace std;
int main()
{
    vector<int> example;           //Vector to store integers
    example.push_back(3);         //Add 3 onto the vector
    example.push_back(10);        //Add 10 to the end
    example.push_back(33);        //Add 33 to the end
    for(int x=0; x<example.size(); x++)
    {
        cout<<example[x]<<" ";    //Should output: 3 10 33
    }
    if(!example.empty())         //Checks if empty
        example.clear();         //Clears vector
    vector<int> another_vector;   //Creates another vector to store integers
    another_vector.push_back(10); //Adds to end of vector
    example.push_back(10);        //Same
    if(example==another_vector)  //To show testing equality
    {
        example.push_back(20);
    }
    for(int y=0; y<example.size(); y++)
    {
        cout<<example[y]<<" ";    //Should output 10 20
    }
    return 0;
}
```

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // create a vector to store int
    vector<int> vec;
    int i;

    // display the original size of vec
    cout << "vector size = " << vec.size() << endl;

    // push 5 values into the vector
    for(i = 0; i < 5; i++){
        vec.push_back(i);
    }

    // display extended size of vec
    cout << "extended vector size = " << vec.size() << endl;

    // access 5 values from the vector
    for(i = 0; i < 5; i++){
        cout << "value of vec [" << i << "] = " << vec[i] << endl;
    }

    // use iterator to access the values
    vector<int>::iterator v = vec.begin();
    while( v != vec.end()) {
        cout << "value of v = " << *v << endl;
        v++;
    }

    return 0;
}
```



```
vector size = 0
extended vector size = 5
value of vec [0] = 0
value of vec [1] = 1
value of vec [2] = 2
value of vec [3] = 3
value of vec [4] = 4
value of v = 0
value of v = 1
value of v = 2
value of v = 3
value of v = 4
```

Η κλάση `list`

- Είναι υλοποιημένη σαν μια **διπλή συνδεδεμένη λίστα**
- Διαφέρει από τη `vector` στον τρόπο υλοποίησης (εσωτερικής αναπαράστασης)
 - Στη `vector` είναι δαπανηρές οι εισαγωγές στη μέση του διανύσματος αλλά είναι πολύ γρήγορη η τυχαία πρόσβαση σε οποιοδήποτε στοιχείο
 - Η `list` παρέχει γρήγορες εισαγωγές στοιχείων σε οποιαδήποτε θέση αλλά αργή τυχαία πρόσβαση (γιατί οι μεταβάσεις μεταξύ στοιχείων είναι διαδοχικές).
 - Χρησιμοποιείται με εξίσου απλό τρόπο όπως και η `vector`


```
std::list<int> integer_list;
integer_list.push_front(1);
integer_list.push_front(2);
```

Φτιάχνει λίστα με το στοιχείο 2 ακολουθούμενο από το στοιχείο 1

```
instance_name.reverse();
```

Αντιστρέφει τη σειρά των στοιχείων

```
instance_name.sort();
```

Ταξινομεί τα στοιχεία της λίστας

```
std::list<int> integer_list;
\\ bad idea
if(integer_list.size() == 0)
    ...
\\ good idea
if(integer_list.empty())
    ...

integer_list.clear();
\\ guaranteed to be empty now
```

```
std::list<int> int_list;
int_list.push_back(1);
int_list.push_back(1);
int_list.push_back(8);
int_list.push_back(9);
int_list.push_back(7);
int_list.push_back(8);
int_list.push_back(2);
int_list.push_back(3);
int_list.push_back(3);
int_list.sort();
int_list.unique();
```




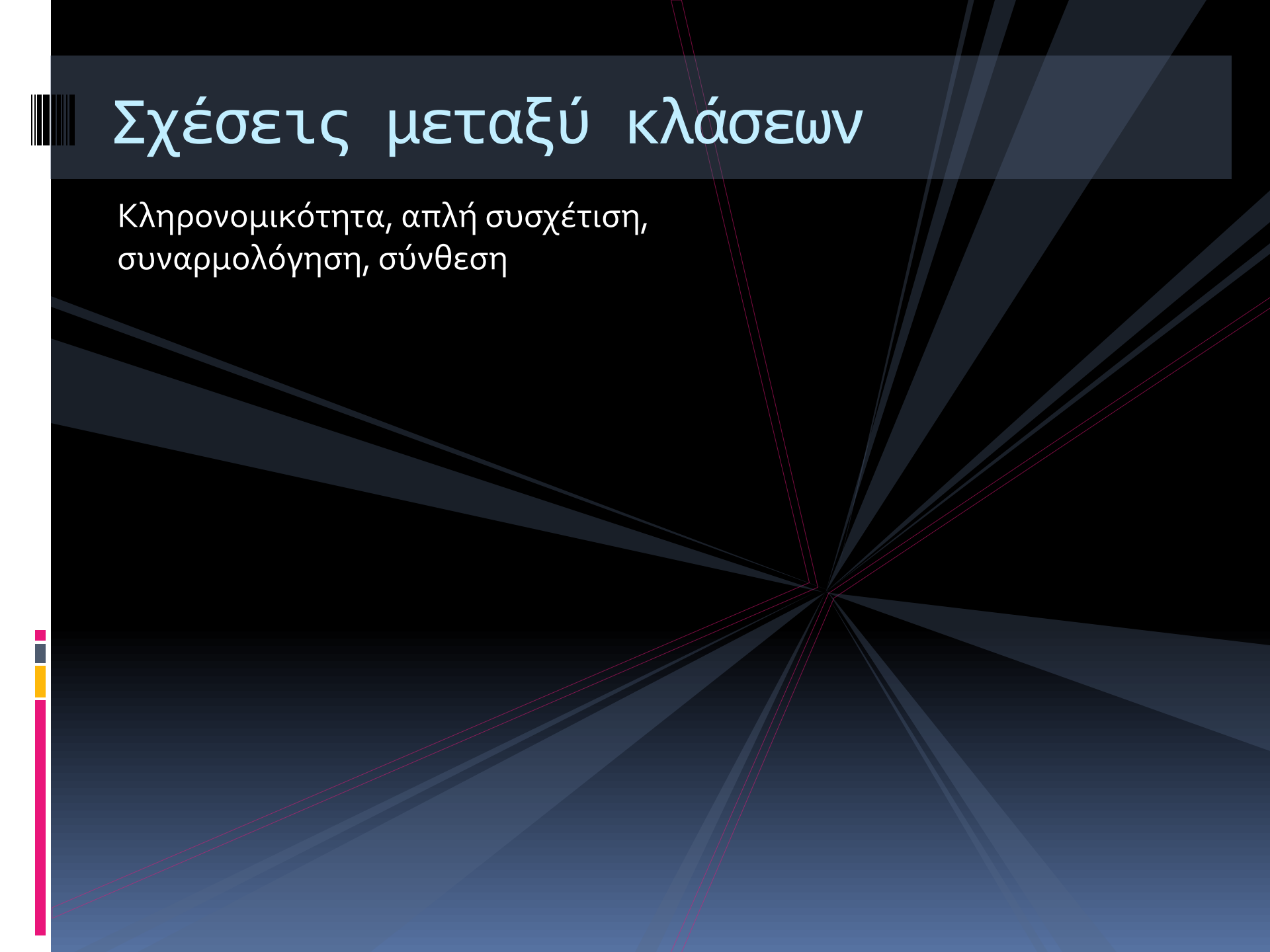
1 8 9 7 8 2 3

```
for(std::list<int>::iterator list_iter = int_list.begin();
    list_iter != int_list.end(); list_iter++)
{
    std::cout<<*list_iter<<endl;
}
```



Σχέσεις μεταξύ κλάσεων

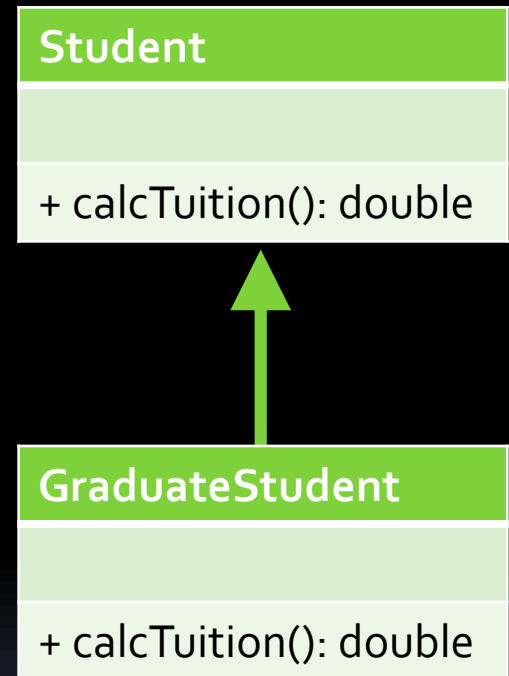
Κληρονομικότητα, απλή συσχέτιση,
συναρμολόγηση, σύνθεση



(Απλή) Κληρονομικότητα

```
class Student
{
    public:
        double calcTuition();
};
class GraduateStudent : public Student
{
    public:
        double calcTuition();
};

int main(int argc, char* pArgs[])
{
    Student s;
    GraduateStudent gs;
    s.calcTuition(); //calls Student::calcTuition()
    gs.calcTuition(); //calls GraduateStudent::calcTuition()
    return 0;
}
```



Στατική διασύνδεση

```
// OverloadOverride - demonstrate when a function is
// overloaded at compile time vs. overridden at runtime
//
//
#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

class Student
{
public:
    // uncomment one or the other of the next
    // two lines; one binds calcTuition() early and
    // the other late
    void calcTuition()
    {
        cout << "We're in Student::calcTuition" << endl;
    }
};

class GraduateStudent : public Student
{
public:
    void calcTuition()
    {
        cout<<"We're in GraduateStudent::calcTuition"<<endl;
    }
};

void fn(Student& x)
{
    x.calcTuition(); // which calcTuition()?
}

int main(int nNumberOfArgs, char* pszArgs[])
{
    // pass a base class object to function
    // (to match the declaration)
    Student s;
    fn(s);

    // pass a specialization of the base class instead
    GraduateStudent gs;
    fn(gs);
}
```

We're in Student::calcTuition
We're in Student::calcTuition
Press Enter to continue...



Δυναμική διασύνδεση

```
We're in Student::calcTuition
We're in GraduateStudent::calcTuition
Press Enter to continue...
```

```
// OverloadOverride - demonstrate when a function is
// overloaded at compile time vs. overridden at runtime
//
//
#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

class Student
{
public:
    // uncomment one or the other of the next
    // two lines; one binds calcTuition() early and
    // the other late
    virtual void calcTuition()
    {
        cout << "We're in Student::calcTuition" << endl;
    }
};

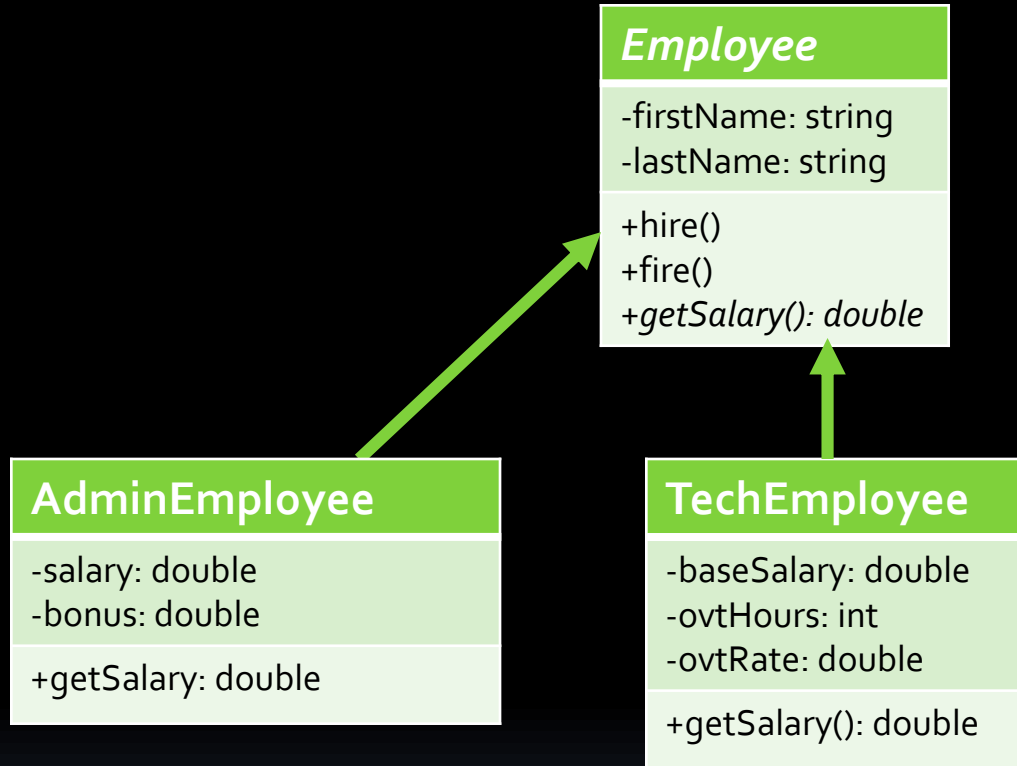
class GraduateStudent : public Student
{
public:
    void calcTuition()
    {
        cout<<"We're in GraduateStudent::calcTuition"<<endl;
    }
};

void fn(Student& x)
{
    x.calcTuition(); // which calcTuition()?
}


int main(int nNumberOfArgs, char* pszArgs[])
{
    // pass a base class object to function
    // (to match the declaration)
    Student s;
    fn(s);

    // pass a specialization of the base class instead
    GraduateStudent gs;
    fn(gs);
}
```

Αφηρημένες μέθοδοι και κλάσεις



```
class Employee
{
public:
    Employee();
    ~Employee();
    void hire();
    void fire();
    virtual double getSalary()=0;
private:
    string firstName;
    string lastName;
};
```



Πολλαπλή κληρονομικότητα, Συσχετίσεις μεταξύ κλάσεων

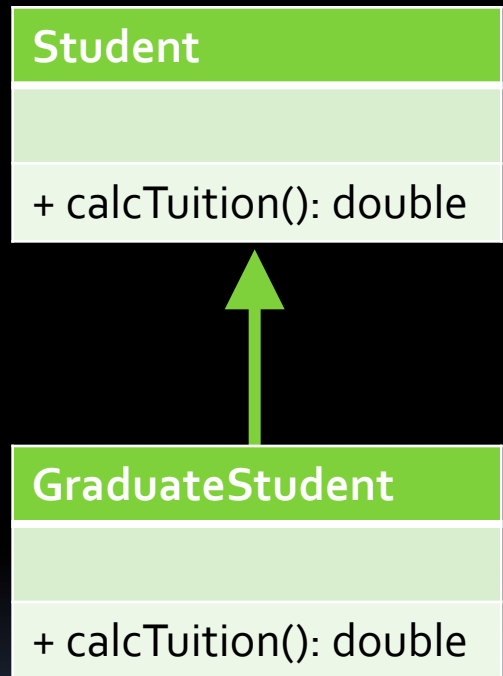
ΟΝΤΟΚΕΝΤΡΙΚΟΣ ΠΡΟΓΡ/ΣΜΟΣ C++

Μ. Ρήγκου (rigou@ceid.upatras.gr)

(Απλή) Κληρονομικότητα

```
class Student
{
    public:
        double calcTuition();
};
class GraduateStudent : public Student
{
    public:
        double calcTuition();
};

int main(int argc, char* pArgs[])
{
    Student s;
    GraduateStudent gs;
    s.calcTuition(); //calls Student::calcTuition()
    gs.calcTuition(); //calls GraduateStudent::calcTuition()
    return 0;
}
```



Στατική διασύνδεση

```
// OverloadOverride - demonstrate when a function is
// overloaded at compile time vs. overridden at runtime
//
#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

class Student
{
public:
    // uncomment one or the other of the next
    // two lines; one binds calcTuition() early and
    // the other late
    void calcTuition()
    {
        cout << "We're in Student::calcTuition" << endl;
    }
};

class GraduateStudent : public Student
{
public:
    void calcTuition()
    {
        cout<<"We're in GraduateStudent::calcTuition"<<endl;
    }
};

void fn(Student& x)
{
    x.calcTuition(); // which calcTuition()?
}

int main(int nNumberOfArgs, char* pszArgs[])
{
    // pass a base class object to function
    // (to match the declaration)
    Student s;
    fn(s);

    // pass a specialization of the base class instead
    GraduateStudent gs;
    fn(gs);
}
```

We're in Student::calcTuition
We're in Student::calcTuition
Press Enter to continue...



Δυναμική διασύνδεση

```
We're in Student::calcTuition
We're in GraduateStudent::calcTuition
Press Enter to continue...
```

```
// OverloadOverride - demonstrate when a function is
// overloaded at compile time vs. overridden at runtime
//
#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

class Student
{
public:
    // uncomment one or the other of the next
    // two lines; one binds calcTuition() early and
    // the other late
    virtual void calcTuition()
    {
        cout << "We're in Student::calcTuition" << endl;
    }
};

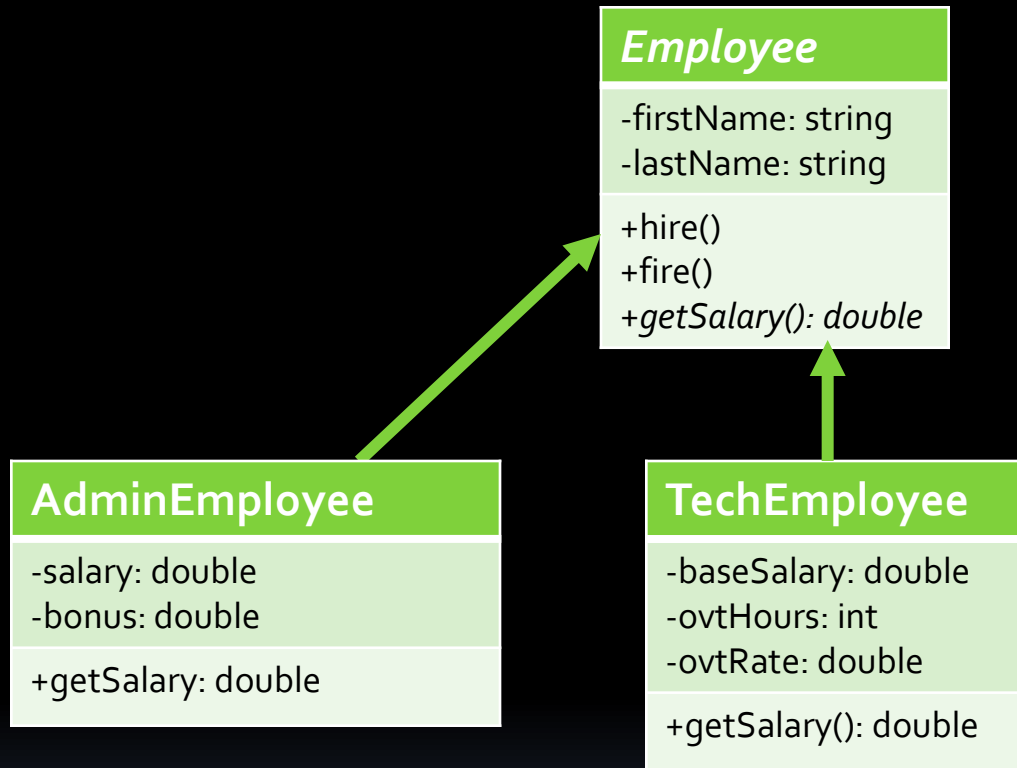
class GraduateStudent : public Student
{
public:
    void calcTuition()
    {
        cout<<"We're in GraduateStudent::calcTuition"<<endl;
    }
};

void fn(Student& x)
{
    x.calcTuition(); // which calcTuition()?
}

int main(int nNumberOfArgs, char* pszArgs[])
{
    // pass a base class object to function
    // (to match the declaration)
    Student s;
    fn(s);

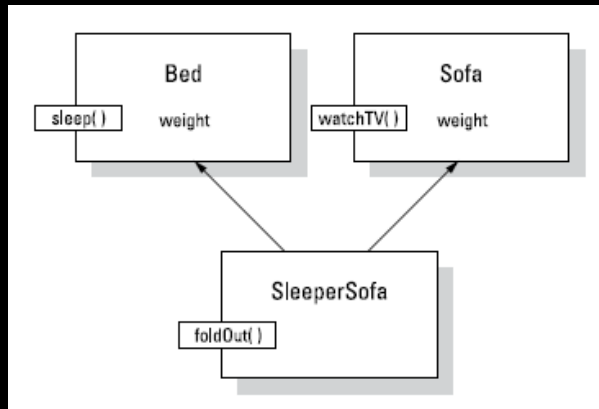
    // pass a specialization of the base class instead
    GraduateStudent gs;
    fn(gs);
}
```

Αφηρημένες μέθοδοι και κλάσεις



```
class Employee
{
public:
    Employee();
    ~Employee();
    void hire();
    void fire();
    virtual double getSalary()=0;
private:
    string firstName;
    string lastName;
};
```

Πολλαπλή κληρονομικότητα



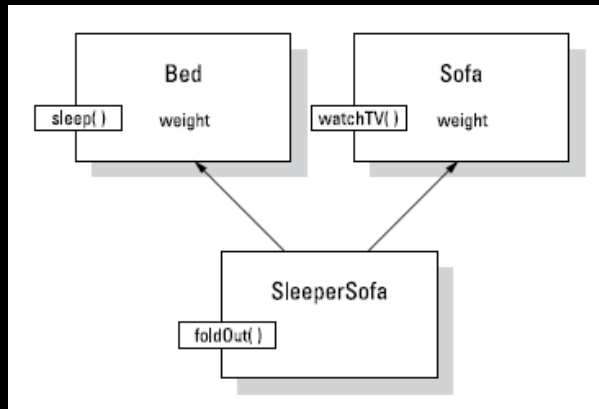
```
// MultipleInheritance - a single class can inherit from
// more than one base class
//
#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

class Bed
{
public:
    Bed(){}
    void sleep(){ cout << "Sleep" << endl; }
    int weight;
};

class Sofa
{
public:
    Sofa(){}
    void watchTV(){ cout << "Watch TV" << endl; }
    int weight;
};

// SleeperSofa - is both a Bed and a Sofa
class SleeperSofa : public Bed, public Sofa
{
public:
    SleeperSofa(){}
    void foldOut(){ cout << "Fold out" << endl; }
};
```

Πολλαπλή κληρονομικότητα



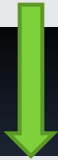
```
int main(int nNumberOfArgs, char* pszArgs[])
{
    SleeperSofa ss;

    // you can watch TV on a sleeper sofa like a sofa...
    ss.watchTV();    // calls Sofa::watchTV()

    //...and then you can fold it out...
    ss.foldOut();    // calls SleeperSofa::foldOut()

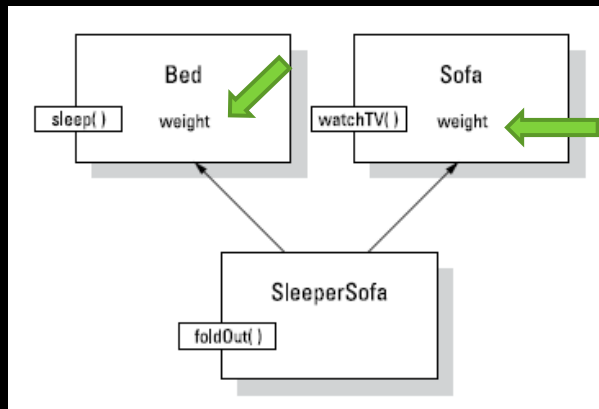
    // ...and sleep on it
    ss.sleep();      // calls Bed::sleep()

    // wait until user is ready before terminating program
    // to allow the user to see the program results
    cout << "Press Enter to continue..." << endl;
    cin.ignore(10, '\n');
    cin.get();
    return 0;
}
```



```
Watch TV
Fold out
Sleep
Press Enter to continue...
```

Πολλαπλή κληρονομικότητα: Ασάφεις



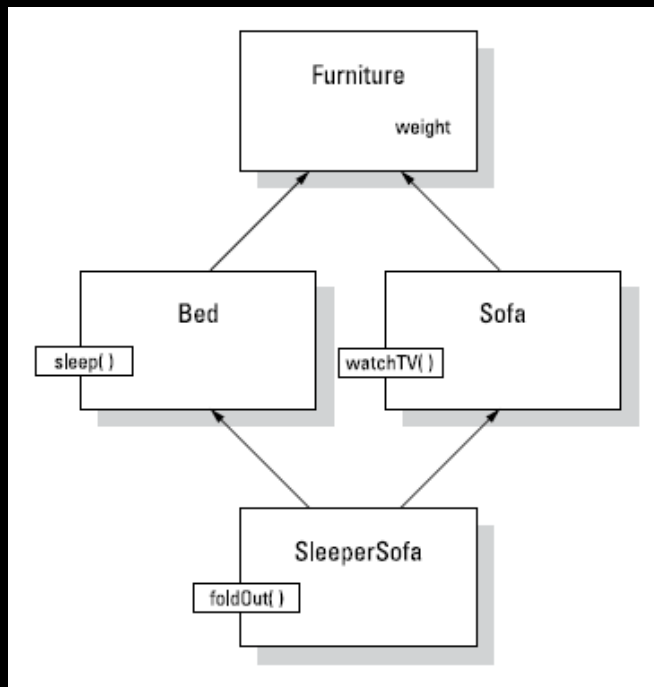
```
#include <iostream>
void fn()
{
    SleeperSofa ss;
    cout << "weight = "
        << ss.weight // illegal - which weight?
        << "\n";
}
```

Πρέπει να το γράψουμε έτσι

```
#include <iostream>
void fn()
{
    SleeperSofa ss;
    cout << "sofa weight = "
        << ss.Sofa::weight // specify which weight
        << "\n";
}
```

Λύνεται το πρόβλημα **ΑΛΛΑ** η συνάρτηση fn() πρέπει να γνωρίζει ότι η κλάση SleeperSofa κληρονομεί το weight από δύο κλάσεις και να ορίσει ποιο από τα δύο weight χρησιμοποιεί. [Κακή αντικειμενοστρεφής σχεδίαση]

Ασάφειες



```
// MultipleInheritanceFactoring - a single class can
//                               inherit from more than one base class
//
#include <cstdio>
#include <cstdlib>
#include <iostream>

using namespace std;

// Furniture - more fundamental concept; this class
//             has "weight" as a property
class Furniture
{
public:
    Furniture(int w) : weight(w) {}
    int weight;
};

class Bed : public Furniture
{
public:
    Bed(int weight) : Furniture(weight) {}
    void sleep(){ cout << "Sleep" << endl; }
};

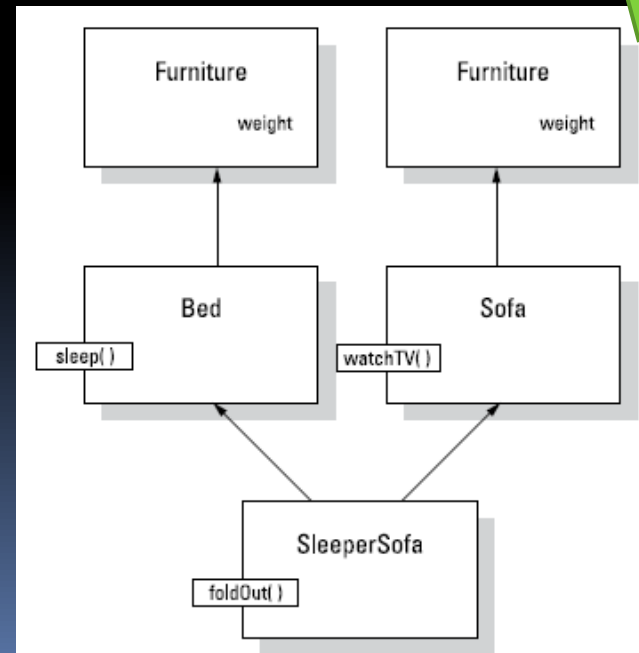
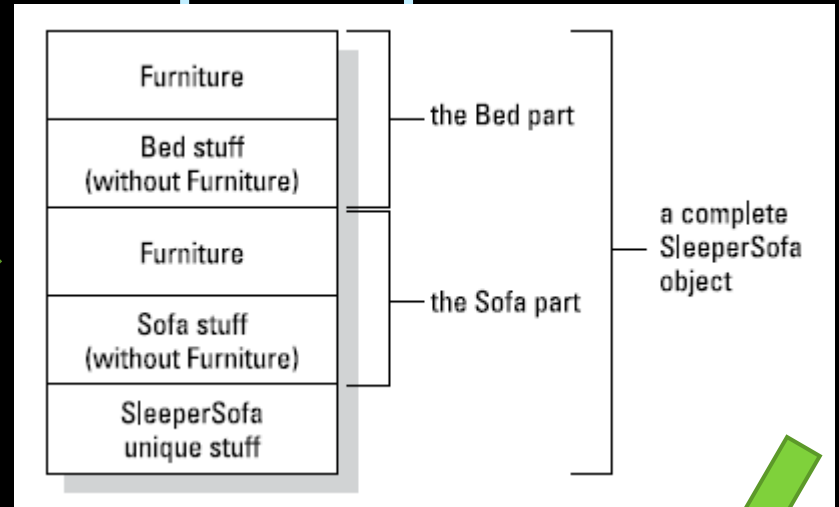
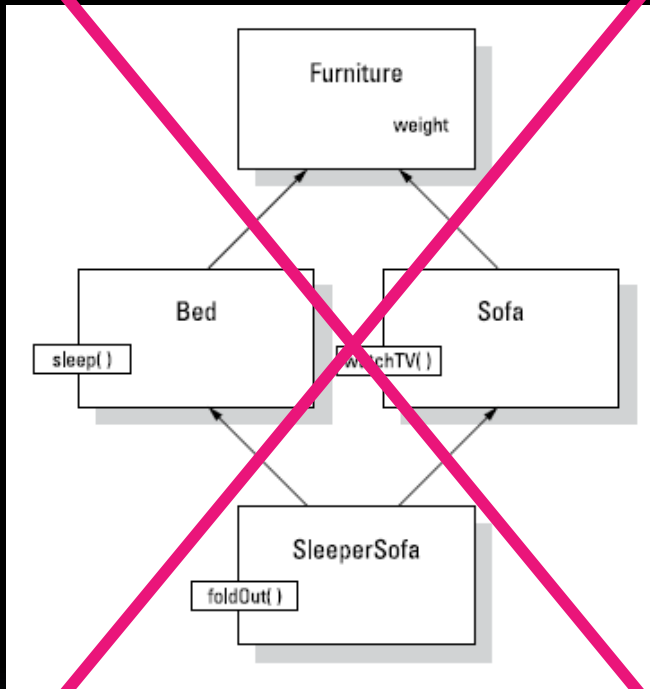
class Sofa : public Furniture
{
public:
    Sofa(int weight) : Furniture(weight) {}
    void watchTV(){ cout << "Watch TV" << endl; }
};

// SleeperSofa - is both a Bed and a Sofa
class SleeperSofa : public Bed, public Sofa
{
public:
    SleeperSofa(int weight) : Bed(weight), Sofa(weight) {}
    void foldOut(){ cout << "Fold out" << endl; }
};
```

```
SleeperSofa ss(10);
```

```
cout << "weight=" << ss.weight << "\n";
```

Στη μνήμη ισοδυναμεί με...



Πολλαπλή κληρονομικότητα: **Ασάφεις**

- Η κλάση SleeperSofa χρειάζεται μόνο ένα αντίγραφο της Furniture και αυτό το ένα αντίγραφο να το μοιράζονται οι κλάσεις Bed και Sofa
- Για να το πετύχουμε αυτό πρέπει να χρησιμοποιήσουμε το keyword **virtual**

Virtual inheritance

```
// Furniture - more fundamental concept; this class
//           has "weight" as a property
class Furniture
{
public:
    Furniture(int w) : weight(w) {}
    int weight;
};
```

```
class Bed : virtual public Furniture
{
public:
    Bed(int w = 0) : Furniture(w) {}
    void sleep() { cout << "Sleep" <<
```

```
class Sofa : virtual public Furniture
{
public:
    Sofa(int w = 0) : Furniture(w) {}
    void watchTV() { cout << "Watch T
```

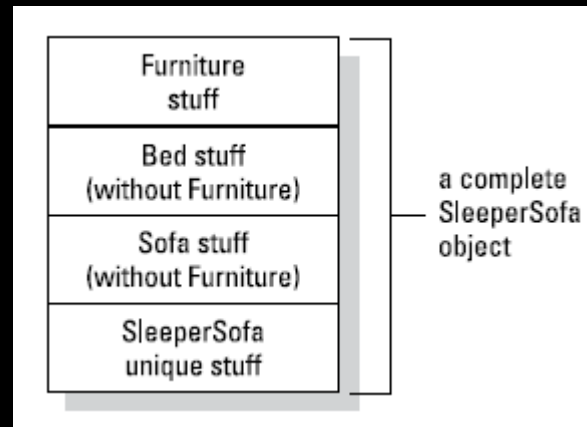
```
// SleeperSofa - is both a Bed and a Sofa
class SleeperSofa : public Bed, public Sofa
{
public:
    SleeperSofa(int w) : Furniture(w) {}
    void foldOut() { cout << "Fold out" << endl; }
};
```

```
int main(int nNumberOfArgs, char* pszArgs[])
{
    SleeperSofa ss(10);

    // the following is no longer ambiguous;
    // there's only one weight shared between Sofa and Bed
    // Furniture::Sofa or a Furniture::Bed?
    cout << "Weight = " << ss.weight << endl;

    // wait until user is ready before terminating program
    // to allow the user to see the program results
    cout << "Press Enter to continue..." << endl;
    cin.ignore(10, '\n');
    cin.get();
    return 0;
}
```

Virtual inheritance



- Η κλάση SleeperSofa κληρονομεί
 - την Furniture,
 - το μέρος της Bed που δεν περιλαμβάνει τα στοιχεία της Furniture
 - και το μέρος της Sofa χωρίς τα στοιχεία της Furniture
- Επιπλέον έχει και τα δικά της μέλη που ορίζει τοπικά

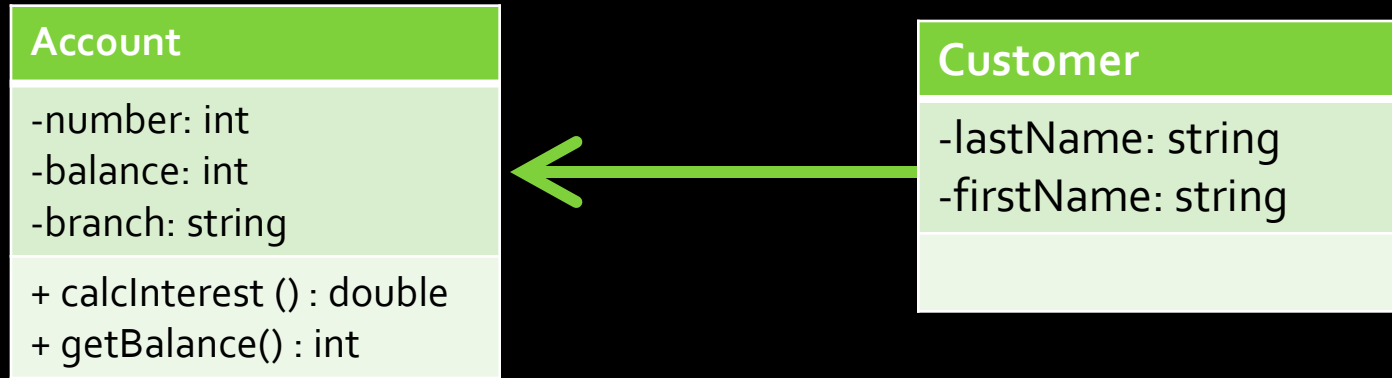


Άλλες συσχετίσεις μεταξύ κλάσεων

Απλή συσχέτιση, συναρμολόγηση/συνάθροιση,
σύνθεση



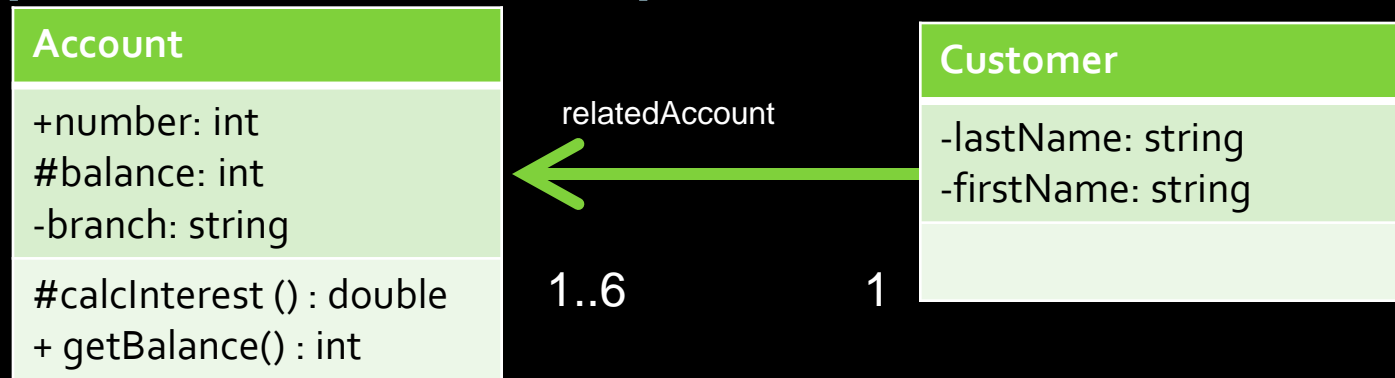
Συσχετίσεις απλής κατεύθυνσης



- Η κλάση **Account** δεν αλλάζει (δεν γνωρίζει τα αντικείμενα της **customer** με τα οποία συνδέεται)
- Στην κλάση **Customer** προσθέτουμε μια ιδιότητα τύπου δείκτη σε αντικείμενο **Account**
 - `Account* the_Account;`
- Και επιπλέον μεθόδους `set_the_Account()` και `get_the_Account()`

```
Account* get_the_Account () const {
return the_Account; }
set_the_Account (Account* value) {
the_Account = value; }
```

Συσχετίσεις απλής κατεύθυνσης με πολλαπλότητα



- Στην κλάση Customer προσθέτουμε μια ιδιότητα τύπου πίνακας με δείκτες σε αντικείμενα Account
 - `Account* related_Account[6];`
- Και επιπλέον μεθόδους `set_related_Account()` και `get_related_Account()`

```
Account * get_related_Account ( int index) const {
return related_Account[index]; }

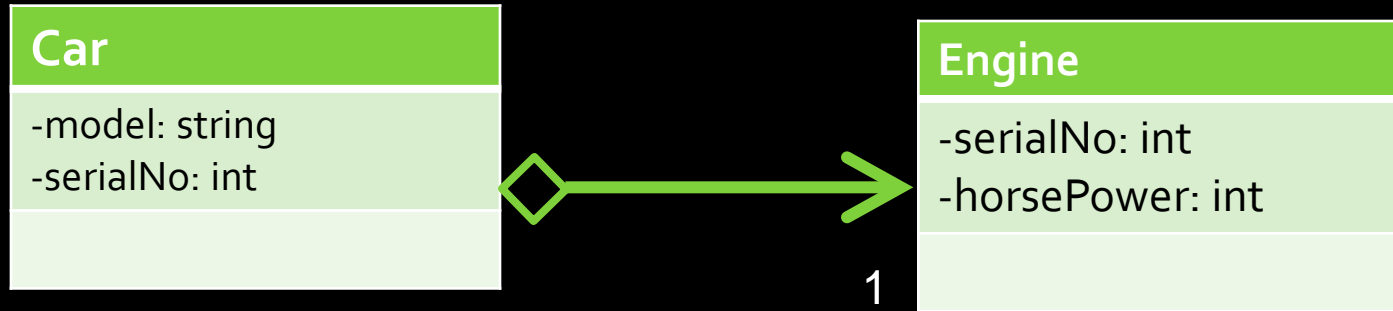
set_related_Account (int index, Account* value) {
related_Account [index] = value; }
```

Συσχετίσεις απλής κατεύθυνσης με πολλαπλότητα

- Άλλο ένα παράδειγμα:

```
class Department
{
    private:
        Professor* members[20];
        int numberOfMembers;
    public:
        Department(Professor* prof) {
            members[0] = prof;
            numberOfMembers = 1;
        }
        void addProfessor(Professor* prof) {
            members[numberOfMembers] = prof;
            numberOfMembers++;
        }
        ~Department() { }
};
```

Συναρμολόγηση (aggregation)



- Συναρμολόγηση: **Κάθε Αυτοκίνητο EXEI (HAS) μια Μηχανή**
 - Αλλά μια μηχανή μπορεί να συνεχίσει να υπάρχει και αφού διαγραφεί το αυτοκίνητο που συνδέεται με αυτή.
- Στην κλάση Car προσθέτουμε μια ιδιότητα τύπου δείκτης σε αντικείμενο Engine
 - `Engine* my_Engine;`
- Και επιπλέον μεθόδους `set_the_Engine ()` και `get_the_Engine ()`

```
Engine * get_the_Engine () const {
return my_Engine; }
set_the_Engine ( Engine* value) {
my_Engine = value; }
```


Συναρμολόγηση (aggregation)

- Ακόμα ένα παράδειγμα:

```
// Aggregation
class Department
{
    private:
        Professor* professor;
    public:
        Department(Professor* prof) : professor(prof) { }
        ~Department() { }
};
```

Συσχετίσεις/συναρμολογήσεις 1-προς-πολλά



```
class Department {  
    public:  
    .....  
    private:  
        vector<Professor*> faculty;  
};
```

Ο ίδιος κώδικας και για απλή συσχέτιση 1-προς-πολλά

Συσχετίσεις/συναρμολογήσεις 1-προς-πολλά

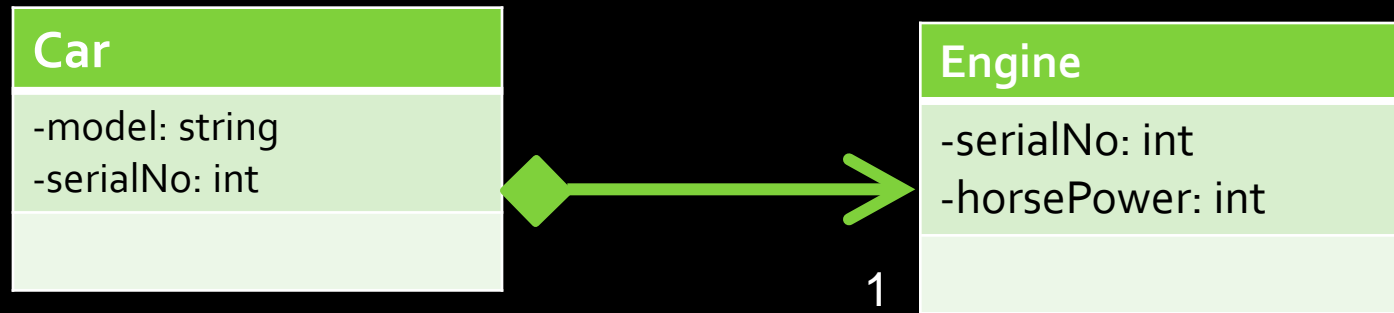


```
Class Department {  
    private:  
        vector<Professor*>* faculty;  
    public:  
        Department (Professor* prof) {  
            faculty = new Vector<Professor*>();  
            faculty->push_back (prof);  
        }  
        void addProfessor (Professor* prof) {  
            faculty->push_back(prof);  
        }  
        ~Department() {delete faculty;}  
};
```

Με δυναμικό τρόπο

Ο ίδιος κώδικας
και για απλή
συσχέτιση 1-
προς-πολλά

Σύνθεση (composition)



- Σύνθεση: **Κάθε Αυτοκίνητο ΕΧΕΙ (HAS) μια Μηχανή**
 - Το αντικείμενο Engine διαγράφεται αυτόματα όταν διαγραφεί το Car που συνδέεται με αυτό
- Στην κλάση Car προσθέτουμε μια ιδιότητα τύπου Engine
 - `Engine my_Engine;`
- Και επιπλέον μεθόδους `set_the_Engine ()` και `get_the_Engine ()`


```
Engine get_the_Engine () const {
return my_Engine; }
set_the_Engine ( Engine value) {
my_Engine = value; }
```

Μπορεί να υλοποιηθεί και με δείκτες...

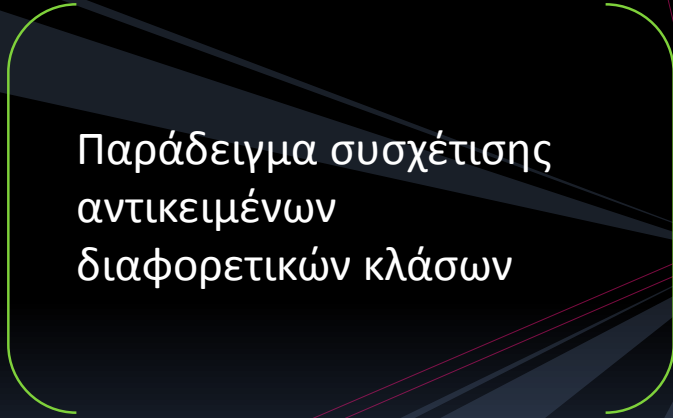


Σύνθεση (composition)

```
// Composition
class Car
{
    private:
        Motor* motor;
    public:
        Car() { motor = new Motor(); }
        ~Car() { delete motor; }
};
```



Επιστροφή στις ασκήσεις με την κλάση Part



Παράδειγμα συσχέτισης
αντικειμένων
διαφορετικών κλάσων

Αντικείμενα ως μέλη κλάσεων

- Άσκηση (class7.cpp)
- Συνεχίστε την υλοποίηση της κλάσης part.
- Να δημιουργηθεί μια νέα κλάση Date της οποίας τα αντικείμενα να αναπαριστούν ημερομηνίες. Τα δεδομένα που χαρακτηρίζουν μια ημερομηνία είναι ο μήνας, η μέρα και ο χρόνος.
- Να δημιουργηθεί νέο ιδιωτικό μέλος της κλάσης part το οποίο να είναι τύπου Date και θα εκφράζει πότε έφτασε στην εταιρία το συγκεκριμένο ανταλλακτικό (arrival_date) και να γίνουν οι απαραίτητες αλλαγές στις συναρτήσεις constructors και στη συνάρτηση show_part για το χειρισμό του.
- Να δημιουργηθεί ένα αντικείμενο τύπου Date με όνομα "d" το οποίο θα αναπαριστά μια ημερομηνία δίνοντας τιμές στα δεδομένα του (month=5,day=11,year=2015).
- Να δημιουργηθεί ένα αντικείμενο τύπου part με όνομα "gear" το οποίο θα αναπαριστά μια ταχύτητα δίνοντας τιμές στα δεδομένα του (modelNumber=2020,partNumber=2025, cost=300.0,company=2,arrival_date=11/5/2015 δηλαδή arrival_day=d).
- Εκτύπωση τιμών δεδομένων αντικειμένου gear.
- Να δημιουργηθεί ένα αντικείμενο τύπου part με όνομα "door".
- Εκτύπωση τιμών δεδομένων αντικειμένου door.



OO και UML

ΟΝΤΟΚΕΝΤΡΙΚΟΣ ΠΡΟΓΡ/ΣΜΟΣ

Μ. Ρήγκου (rigou@ceid.upatras.gr)



Ένα πλήρες παράδειγμα σε Java

Με επισκόπηση κάποιων διαγραμμάτων UML



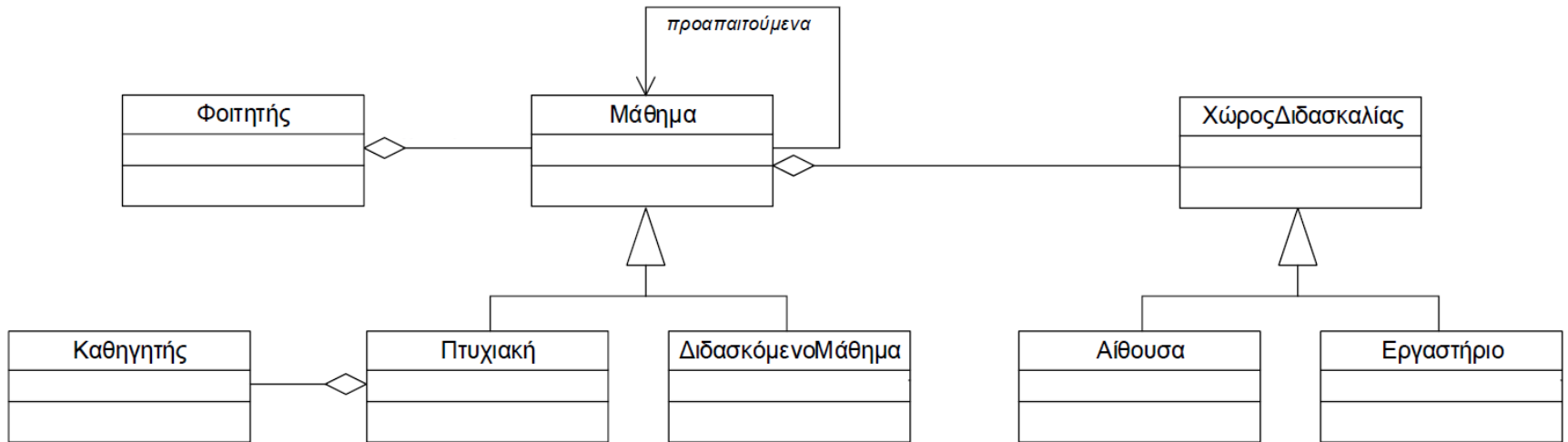
Σενάριο συστήματος: Εγγραφή Φοιτητή σε Μάθημα

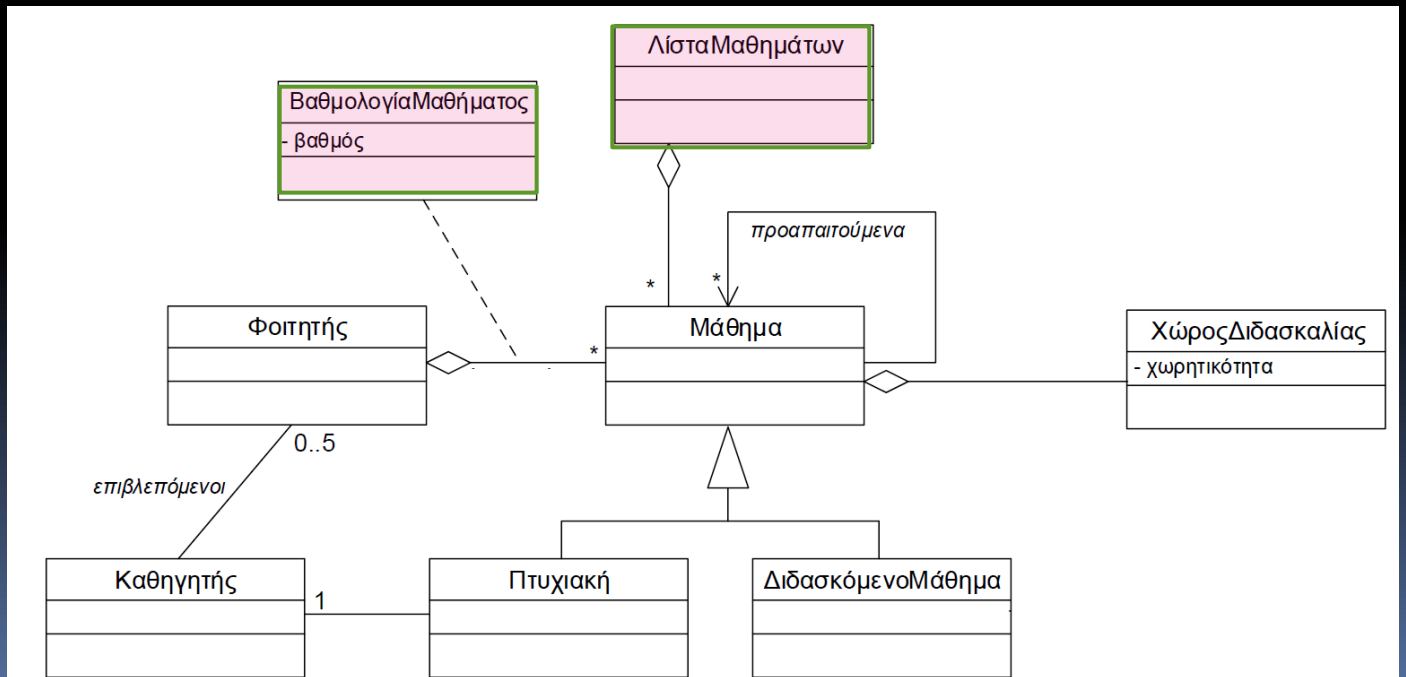
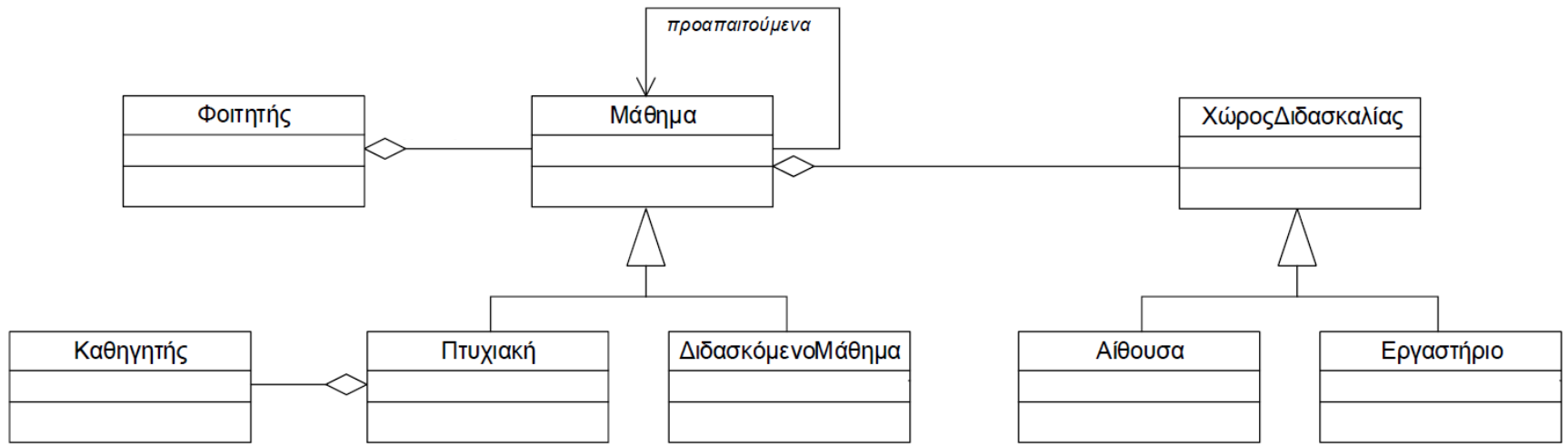
- Για να μπορέσει ο φοιτητής να εγγραφεί σε ένα μάθημα θα πρέπει να συντρέχουν κάποιες προϋποθέσεις:
 - Ο φοιτητής θα πρέπει να έχει περάσει επιτυχώς τα **προαπαιτούμενα** μαθήματα για το μάθημα που επιλέγει
 - Κάθε μάθημα πραγματοποιείται σε μια αίθουσα ή σε ένα εργαστήριο με καθορισμένη χωρητικότητα. Επομένως θα πρέπει να υπάρχει **διαθέσιμος χώρος** στην αντίστοιχη αίθουσα ή το εργαστήριο
 - Αν το επιλεγμένο μάθημα είναι η πτυχιακή εργασία σε έναν καθηγητή θα πρέπει ο αντίστοιχος καθηγητής να έχει **περιθώριο επίβλεψης** (κάθε καθηγητής μπορεί να επιβλέψει έως και 5 πτυχιακές εργασίες).
- Αν οποιαδήποτε συνθήκη δεν πληρείται θα εμφανίζεται κατάλληλο μήνυμα και ο φοιτητής δεν θα εγγράφεται στο μάθημα

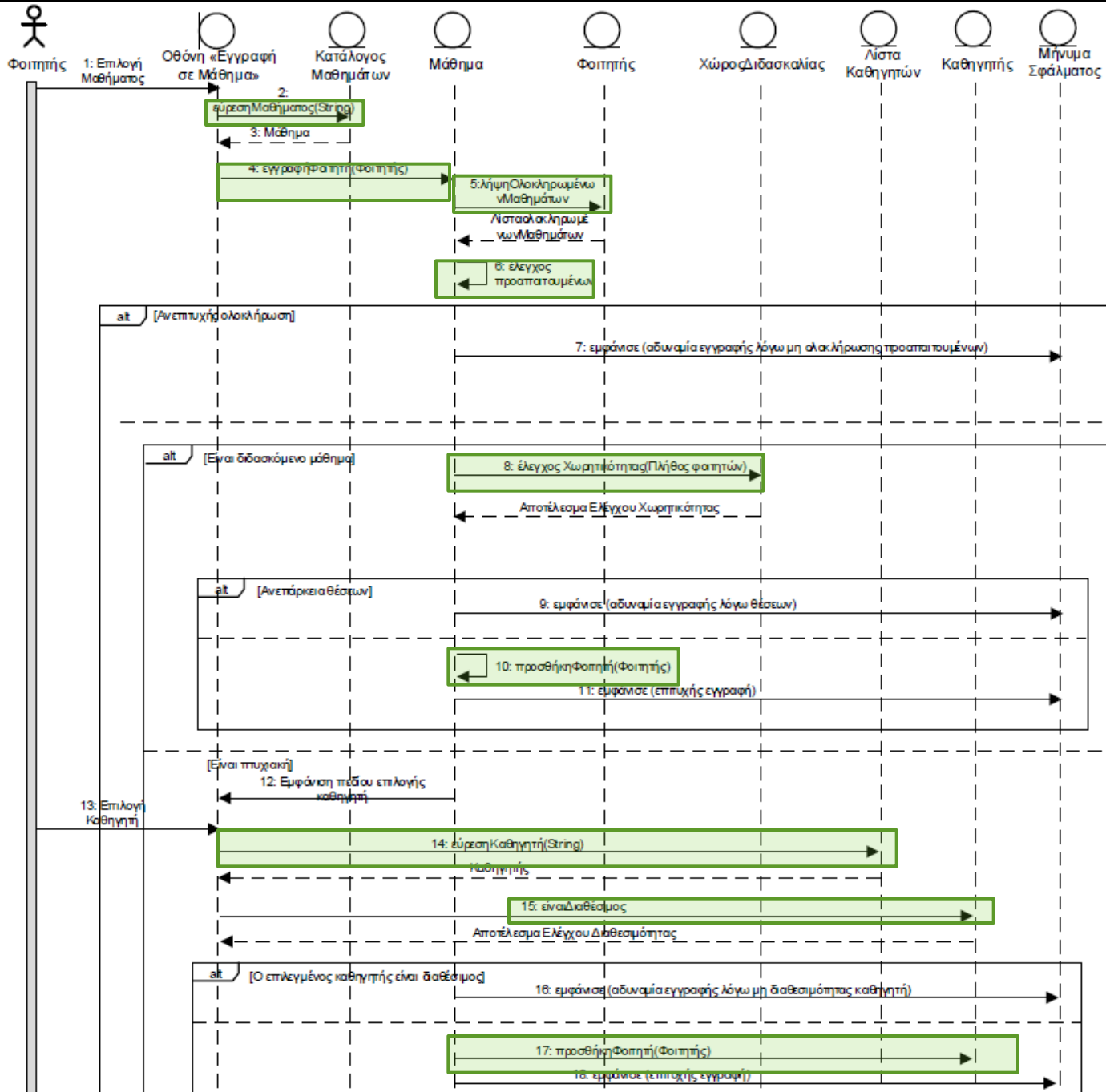
Σενάριο συστήματος: Εγγραφή Φοιτητή σε Μάθημα

- Για να μπορέσει ο φοιτητής να εγγραφεί σε ένα μάθημα θα πρέπει να συντρέχουν κάποιες προϋποθέσεις:
 - Ο φοιτητής θα πρέπει να έχει περάσει επιτυχώς τα προαπαιτούμενα μαθήματα για το μάθημα που επιλέγει. Τα μαθήματα είναι είτε διδασκόμενα μαθήματα είτε πτυχιακές εργασίες.
 - Κάθε μάθημα πραγματοποιείται σε ένα χώρο διδασκαλίας (σε μια αίθουσα ή σε ένα εργαστήριο) με καθορισμένη χωρητικότητα. Επομένως θα πρέπει να υπάρχει διαθέσιμος χώρος
 - Αν το επιλεγμένο μάθημα είναι η πτυχιακή εργασία σε έναν καθηγητή θα πρέπει ο αντίστοιχος καθηγητής να έχει περιθώριο επίβλεψης (κάθε καθηγητής μπορεί να επιβλέψει έως και 5 πτυχιακές εργασίες).
- Αν οποιαδήποτε συνθήκη δεν πληρείται θα εμφανίζεται κατάλληλο μήνυμα και ο φοιτητής δεν θα εγγράφεται στο μάθημα

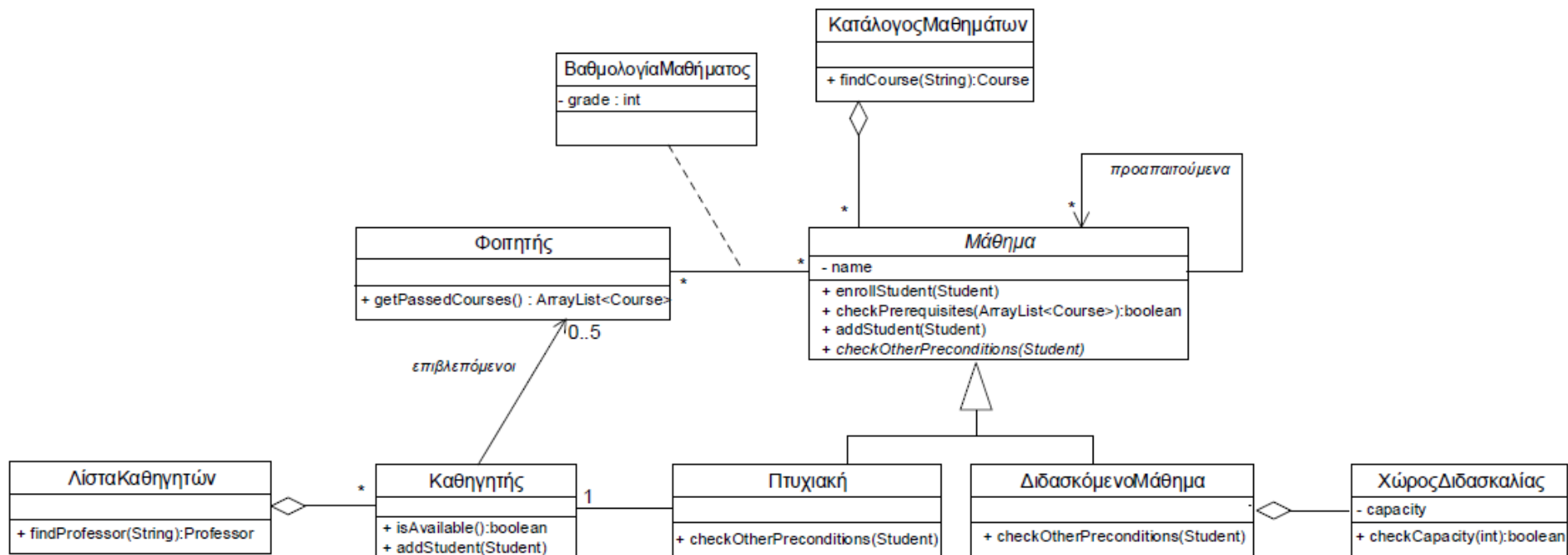
Αρχικό διάγραμμα κλάσεων UML







Πλήρες διάγραμμα κλάσεων



Από το διάγραμμα κλάσεων στον κώδικα

- Στον κώδικα των κλάσεων έχουν προστεθεί επιπλέον μέθοδοι (αυτών που προκύπτουν από το διάγραμμα ακολουθίας) για να ελεγχθεί η λειτουργικότητα (π.χ. η `AddCourse` στην κλάση `CourseList`) και επιπλέον ιδιότητες που θα προέκυπταν εάν είχε αναλυθεί ολόκληρο το σύστημα
- Οι κλάσεις `CourseList` και `ProfessorList` έχουν δηλωθεί με στατικές μεθόδους για ευκολία κλήσης τους (δεν χρειάζεται να δημιουργήσουμε αντικείμενα, τις καλούμε επί των κλάσεων)

Η κλάση Professor

```
import java.util.ArrayList;
public class Professor {
    private String name;
    private int capacity;
    private ArrayList<Student> students = new ArrayList<Student>();

    public Professor(String name, int capacity) {
        this.name = name;
        this.capacity = capacity;
    }

    public void addStudent(Student s) {
        students.add(s);
    }

    public boolean isAvailable() {
        if(students.size() < capacity)
            return true;
        return false;
    }

    public String getName() {
        return name;
    }
}
```

Professor

-name: String

-capacity: int

+isAvailable(): boolean

+addStudent(Student)

Η κλάση ProfessorList

```
import java.util.ArrayList;
public class ProfessorList {
    private static ArrayList<Professor> professors = new
    ArrayList<Professor>();

    public static void addProfessor(Professor aProfessor) {
        professors.add(aProfessor);    }

    public static Professor findProfessor(String name) {
        for(Professor professor: professors)
            if(professor.getName().equals(name))
                return professor;
        return null;    }
}
```

ProfessorList

+findProfessor(String):Professor
+addProfessor(Professor)

Η κλάση CourseList

```
import java.util.ArrayList;
```

```
public class CourseList {
```

```
    private static ArrayList<Course> courses = new  
    ArrayList<Course>();
```

```
    public static void addCourse(Course aCourse) {  
        courses.add(aCourse);  
    }
```

```
    public static Course findCourse(String name) {  
        for(Course course: courses)  
            if(course.getName().equals(name))  
                return course;  
        return null;  
    }  
}
```

CourseList

+findCourse(String):Course

Η κλάση Room

```
public class Room {  
  
    private String name;  
    private int capacity;  
  
    public Room(String name, int capacity) {  
        this.name = name;  
        this.capacity = capacity;  
    }  
  
    public boolean checkCapacity(int students) {  
        if(students > capacity)  
            return false;  
        return true;  
    }  
}
```

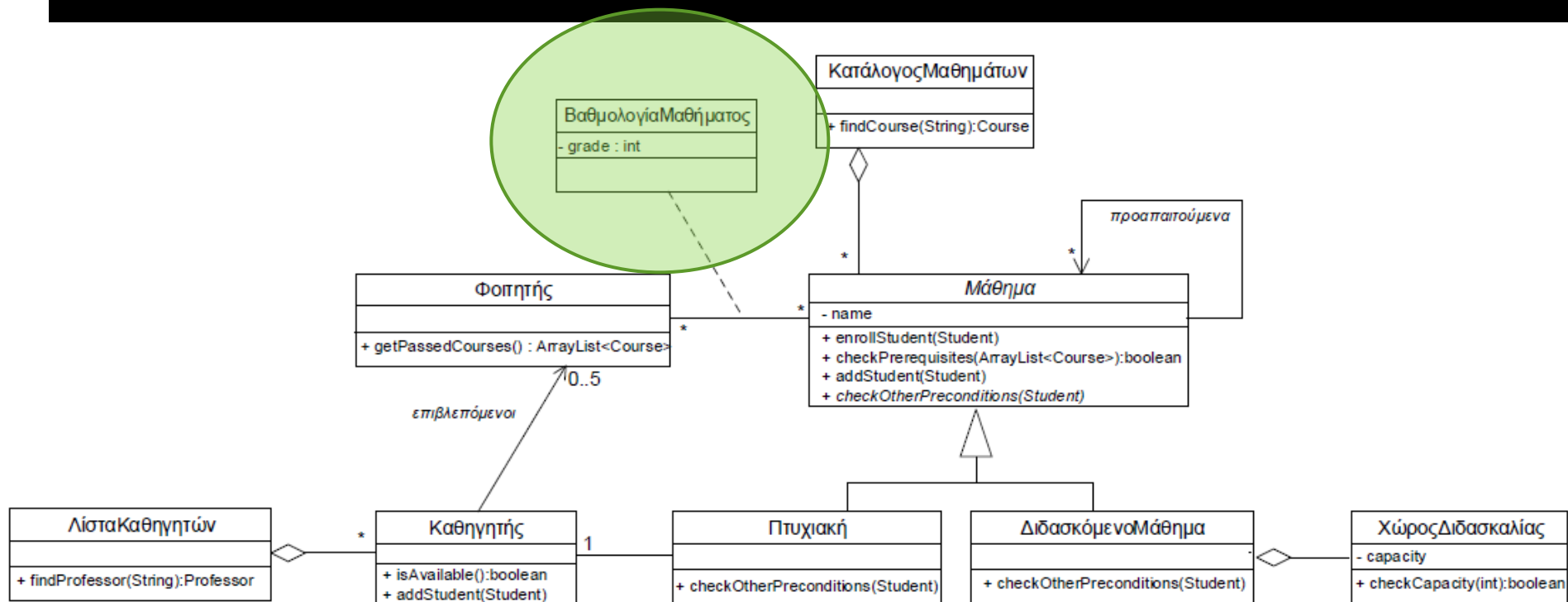
Room

-name:String

-capacity:int

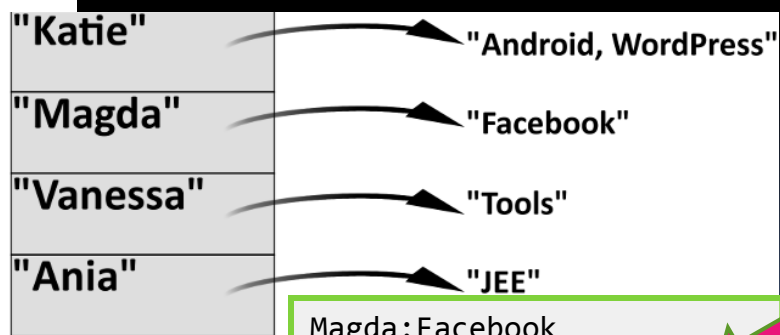
+CheckCapacity (int):boolean

Πλήρες διάγραμμα κλάσεων



HashMap

- Μια δομή που αποθηκεύει ζευγάρια <κλειδί, τιμή> (<key, value>)
- Κάθε κλειδί εμφανίζεται μόνο 1 φορά (κάνει overwrite αν προσθέσουμε ζευγάρι με το ίδιο κλειδί ξανά)
- Μπορεί να περιέχει ένα null key και πολλαπλές null values
- Δεν εφαρμόζει κάποια διάταξη



Magda:Facebook
Vanessa:Tools
Ania:JEE
Katie:Android, WordPress

HashMap - keys

```
// Create the HashMap
HashMap<String,String> hm = new HashMap<String, String>();

// Put data
hm.put("Katie", "Android, WordPress");
hm.put("Magda", "Facebook");
hm.put("Vanessa", "Tools");
hm.put("Ania", "Java");
hm.put("Ania", "JEE"); // !! Put another data under the same
// key, the old value is overridden

// HashMap iteration
for (String key: hm.keySet())
    System.out.println(key+" "+hm.get(key));
```

HashMap

Μέθοδοι κλάσης HashMap	Περιγραφή
<code>void clear()</code>	Διαγράφει όλα τα στοιχεία του map
<code>boolean containsKey(Object key)</code>	True αν υπάρχει στα στοιχεία του map ζευγάρι με το συγκεκριμένο κλειδί
<code>boolean containsValue(Object value)</code>	True αν υπάρχει στα στοιχεία του map ζευγάρι με τη συγκεκριμένη τιμή
<code>Object get(Object key)</code>	Επιστρέφει την τιμή που αντιστοιχεί στο συγκεκριμένο κλειδί ή null
<code>Set keySet()</code>	Επιστρέφει το σύνολο των κλειδιών του map
<code>Object put(Object key, Object value)</code>	Προσθέτει το ζευγάρι <key, value>
<code>int size()</code>	Επιστρέφει το πλήθος των ζευγαριών στο Map

Η κλάση Student

```
import java.util.ArrayList;
import java.util.HashMap;

public class Student {
    private String name;
    //HashMap that contains <Course,grade> pairs
    private HashMap<Course, Integer> courses = new HashMap<Course,
    Integer>();

    public Student(String name) { this.name = name; }

    public String getName() { return name; }

    public void storeGrade(Course aCourse, int grade) {
        courses.put(aCourse, grade);
    }

    public ArrayList<Course> getPassedCourses() {
        ArrayList<Course> passedCourses = new ArrayList<Course>();
        for(Course course: courses.keySet()) {
            if(courses.get(course) >= 5)
                passedCourses.add(course);
        }
        return passedCourses;
    }
}
```

Student

-name

+getPassedCourses()

+storeGrade()

Η κλάση Course (1)

```
import java.util.ArrayList;
public abstract class Course {
    private String name;
    private ArrayList<Course> preRequisites = new ArrayList<Course>();
    protected ArrayList<Student> enrolledStudents = new
    ArrayList<Student>();

    public Course(String name) {this.name = name;}
    public String getName() {return name;}
    public abstract void checkOtherPreconditions(Student aStudent);

    public void enrollStudent(Student aStudent) {
        ArrayList<Course> passedCourses = aStudent.getPassedCourses();
        if(!checkPrerequisites(passedCourses))
            System.out.println("Prerequisites not Passed");
        else {
            checkOtherPreconditions(aStudent);    }
    }
}
```

Η κλάση Course (2)

```
public void addPrerequisite(Course aCourse) {  
    preRequisites.add(aCourse);    }
```

```
public boolean checkPrerequisites(ArrayList<Course> passedCourses) {  
    for(Course preRequisite: preRequisites) {  
        boolean found = false;  
        for(Course passedCourse : passedCourses)  
            if(preRequisite.getName().equals(passedCourse.getName()))  
                found = true;  
        if(!found)  
            return false;    }  
    return true;    }
```

```
public void addStudent(Student aStudent) {  
    enrolledStudents.add(aStudent);  
    System.out.println("Student: " + aStudent.getName() + " has been  
successfully enrolled");    } }
```

Course

-name: String

+enrollStudent(Student)

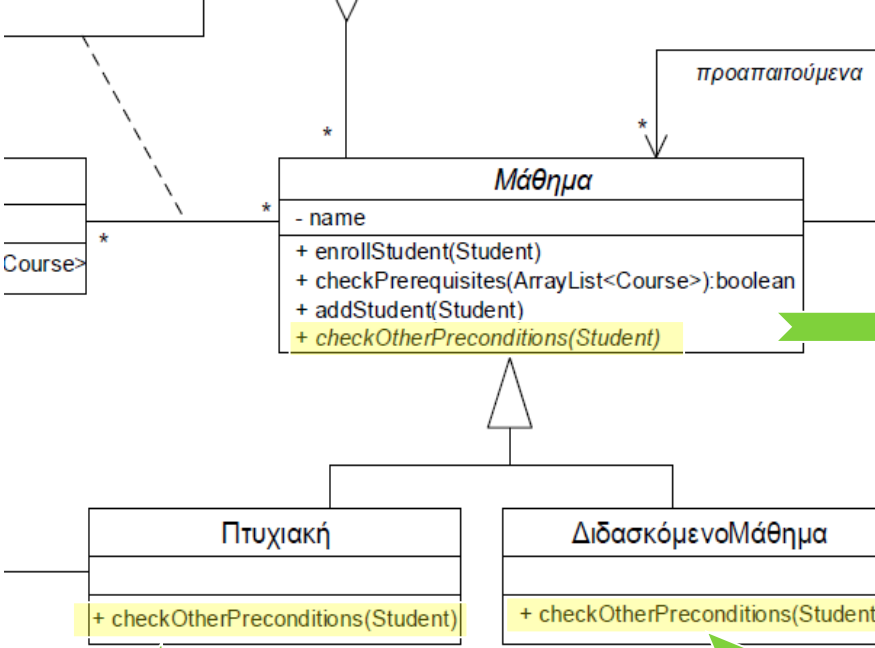
+addPrerequisite(Course)

+checkPrerequisites(ArrayList<Course>):boolean

+addStudent(Student)

+*CheckOtherPreconditions(Student)*

Πολυμορφισμός



```
public abstract void checkOtherPreconditions(Student aStudent);
```

```
public void checkOtherPreconditions(Student aStudent) {
    boolean capacityCheck = room.checkCapacity(enrolledStudents.size()+1);
    if(!capacityCheck)
        System.out.println("No available space in the associated room");
    else
        addStudent(aStudent);
}
```

```
public void checkOtherPreconditions(Student aStudent) {

    Scanner in = new Scanner(System.in);
    System.out.println("Enter Professor Name: ");
    String name = in.nextLine();

    Professor selectedProfessor = ProfessorList.findProfessor(name);

    boolean availabilityCheck = selectedProfessor.isAvailable();
    if(!availabilityCheck)
        System.out.println("The corresponding professor is not available");
    else {
        selectedProfessor.addStudent(aStudent);
        System.out.println("Student: " + aStudent.getName() + " has been successfully enrolled");
    }
}
```

Πολυμορφισμός

- Για τη διαφοροποίηση των πτυχιακών από τα διδασκόμενα μαθήματα, μπορούμε να εκμεταλλευτούμε τη δυνατότητα του πολυμορφισμού και να αποφύγουμε το «ερώτημα» αν ένα αντικείμενο που έχουμε στα «χέρια μας» είναι πτυχιακή ή διδασκόμενο μάθημα
 - Υλοποίηση αφηρημένης μεθόδου `checkOtherPreconditions` στην υπερκλάση Μάθημα
 - Επικάλυψη της μεθόδου σε κάθε υποκλάση ανάλογα με τις ανάγκες του κάθε είδους μαθήματος στις Πτυχιακή και Διδασκόμενο
 - Κατά την εγγραφή καλείται η αφηρημένη μέθοδος `checkOtherPreconditions`
- Ανάλογα με το είδος του αντικειμένου Μάθημα, κατά τη διάρκεια εκτέλεσης μέσα στην `enrollStudent()` καλείται η κατάλληλη overridden μέθοδος (ένα μήνυμα -> πολλές μορφές: πολυμορφισμός)

Η κλάση TaughtCourse

```
public class TaughtCourse extends Course {  
    public TaughtCourse(String name) {  
        super(name);    }  
    private Room room;  
    public void setRoom(Room aRoom) {  
        room = aRoom;    }  
  
    public void checkOtherPreconditions(Student aStudent) {  
        boolean capacityCheck =  
            room.checkCapacity(enrolledStudents.size()+1);  
        if(!capacityCheck)  
            System.out.println("No available space in the  
associated room");  
        else  
            addStudent(aStudent);    }  
}
```

TaughtCourse

+CheckOtherPreconditions(Student)

```
import java.util.Scanner;
```

```
public class Thesis extends Course {
```

```
    private Professor professor;
```

```
    public Thesis(String name) {
```

```
        super(name);    }
```

```
    public void setProfessor(Professor aProfessor) {
```

```
        professor = aProfessor;    }
```

```
    public void checkOtherPreconditions(Student aStudent) {
```

```
        Scanner in = new Scanner(System.in);
```

```
        System.out.println("Enter Professor Name: ");
```

```
        String name = in.nextLine();
```

```
        Professor selectedProfessor = ProfessorList.findProfessor(name);
```

```
        boolean availabilityCheck = selectedProfessor.isAvailable();
```

```
        if(!availabilityCheck)
```

```
            System.out.println("The corresponding professor is not available");
```

```
        else {
```

```
            selectedProfessor.addStudent(aStudent);
```

```
            System.out.println("Student: " + aStudent.getName() + " has  
been successfully enrolled");    }
```

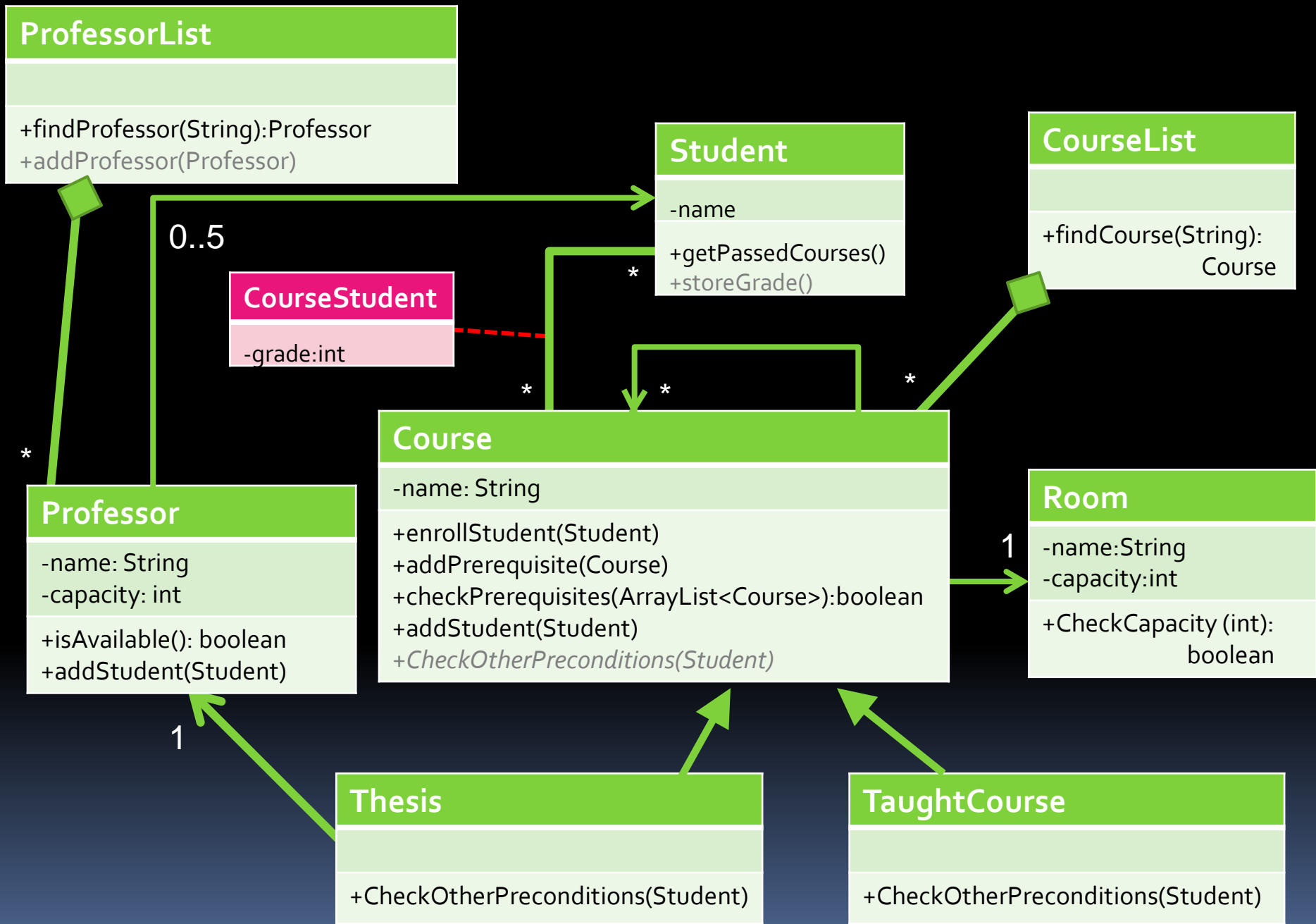
```
    }
```

```
}
```

Η κλάση Thesis

Thesis

+CheckOtherPreconditions(Student)



Main()

```
//Room R1 has capacity equal to 1 only for test purposes
```

```
Room R1 = new Room("LectureHall 3", 1);
```

```
TaughtCourse C1 = new TaughtCourse("Math");
```

```
TaughtCourse C2 = new TaughtCourse("Databases");
```

```
TaughtCourse C3 = new TaughtCourse("Java");
```

```
TaughtCourse C4 = new TaughtCourse("Graphics");
```

```
Professor P1 = new Professor("Johnson", 0);
```

```
Professor P2 = new Professor("Roberts", 5);
```

```
Thesis C5 = new Thesis("Design Patterns");
```

```
Thesis C6 = new Thesis("Big Data");
```

```
CourseList.addCourse(C1);
```

```
CourseList.addCourse(C2);
```

```
CourseList.addCourse(C3);
```

```
CourseList.addCourse(C4);
```

```
CourseList.addCourse(C5);
```

```
CourseList.addCourse(C6);
```

Main()

```
ProfessorList.addProfessor(P1);
```

```
ProfessorList.addProfessor(P2);
```

```
System.out.println("Java has Math and Databases as prerequisites");
```

```
C3.addPrerequisite(C1);
```

```
C3.addPrerequisite(C2);
```

```
System.out.println("Graphics has only Math as Prerequisite");
```

```
C4.addPrerequisite(C1);
```

```
System.out.println();
```

```
C4.setRoom(R1);
```

```
System.out.println("Graphics is being taught in Room R1");
```

```
System.out.println();
```

```
Student S1 = new Student("George");
```

```
System.out.println("Student George has been created");
```

```
S1.storeGrade(C1, 6);
```

```
System.out.println("George has passed Math");
```

Main()

```
System.out.println("Trying to enroll George to Java...");  
C3.enrollStudent(S1); //unable to  
System.out.println();
```

```
System.out.println("Trying to enroll George to Graphics...");  
C4.enrollStudent(S1); //Student is successfully enrolled  
System.out.println();  
//Attempt to enroll another student  
//in the same course  
//No more place  
Student S2 = new Student("John");  
System.out.println("Student John has been created");
```

```
S2.storeGrade(C1, 8); //John has passed Math
```

```
System.out.println("Trying to enroll John to Graphics...");  
C4.enrollStudent(S2); //No available space  
System.out.println();  
//Student S1 is expected to input "Johnson" who is not  
//available  
System.out.println("Trying to enroll George to professor Johnson for his Thesis...");  
System.out.println("Give the name of George's thesis supervisor: [Please type: Johnson]");  
C5.enrollStudent(S1);  
System.out.println();  
//Student S1 is expected to input "Roberts" who is  
//available  
System.out.println("Trying to enroll George to professor Roberts for his Thesis...");  
System.out.println("Give the name of George's thesis supervisor: [Please type: Roberts]");  
C6.enrollStudent(S1);
```

Παρατηρήσεις

- Για τη σχεδίαση ενός συστήματος υπάρχουν πολλές **εναλλακτικές** που μπορεί να εξασφαλίζουν την ίδια λειτουργικότητα αλλά να διαφέρουν **ποιοτικά**
- Φροντίζουμε να κατανέμουμε τη λειτουργικότητα στις κλάσεις και να μην τη συσσωρεύουμε σε μια «Θεϊκή» κλάση
- Χρησιμοποιήσαμε **πολυμορφισμό** για να αποφύγουμε να ελέγξουμε αν ένα Μάθημα είναι Διδασκόμενο ή Πτυχιακή
- Ο **Κατάλογος Καθηγητών** χρειάζεται επειδή στο σενάριο θεωρούμε ότι ο φοιτητής εισάγει το όνομα του Καθηγητή και πρέπει να εντοπίσουμε το αντίστοιχο αντικείμενο