

Η ΓΛΩΣΣΑ C++

Μάθημα 2:

Εισαγωγή στις Κλάσεις

Δημήτρης Ψούνης



www.psounis.gr



Περιεχόμενα Μαθήματος

A. Θεωρία

1. Κλάσεις

1. Γενικά
2. Ορισμός Κλάσης
3. Δημόσια (public) στοιχεία της κλάσης
4. Ιδιωτικά (private) στοιχεία της κλάσης
5. Παράδειγμα (προδιαγραφές)

2. Περισσότερα για τις κλάσεις

1. Ορισμός Συναρτήσεων έξω από την Κλάση
2. Παρουσίαση Ιδιωτικών – Δημόσιων Μέλων μιας κλάσης
3. Χωρισμός σε Αρχεία

3. Ειδικές Μεθόδοι Κλάσεων

1. Γενικά
2. Κατασκευαστής (constructor)
3. Καταστροφέας (destructor)
4. Ελεγκτές Πρόσβασης (accessors)

B. Ασκήσεις



A. Θεωρία

1. Κλάσεις

1. Γενικά

Μία κλάση:

- Μοντελοποιεί μια ιδέα, ένα πρότυπο, μια κατηγορία οντοτήτων, έναν χαρακτηρισμό.
 - Ασχολούμαστε με το να ορίσουμε «τι είναι» αυτή η κατηγορία οντοτήτων.
 - Προσοχή όμως, ότι αυτή η ιδέα «δεν υπάρχει» στον πραγματικό κόσμο
- Αποτελείται από δεδομένα και συναρτήσεις.
 - Τα δεδομένα καλούνται μέλη της κλάσης.
 - Ορίζουν χαρακτηριστικά της κλάσης.
 - Οι συναρτήσεις λέγονται και μέθοδοι της κλάσης.
 - Ορίζουν τη συμπεριφορά της κλάσης

Ένα αντικείμενο

- Είναι ένα στιγμιότυπο της κλάσης
- Μπορούν να οριστούν οσαδήποτε στιγμιότυπα της κλάσης.
- Είναι κάτι που «υπάρχει» στον πραγματικό κόσμο.

Παράδειγμα 1:

- Ο σκύλος είναι ένα πρότυπο, μια ιδέα.
- Ο Αζόρ, ο σκύλος της Γιάννας, η Λάσι, ο Πίκο είναι αντικείμενα της κλάσης «σκύλος»

Παράδειγμα 2:

- Το τρίγωνο είναι ένα πρότυπο, μια ιδέα.
- Ένα ισόπλευρο τρίγωνο που έχω ζωγραφίσει αυτήν τη στιγμή στο χαρτί μου είναι ένα αντικείμενο της κλάσης «τρίγωνο»

Παράδειγμα 3:

- Ένας μηχανισμός εξόδου της C++ είναι ένα πρότυπο, μια ιδέα.
- Το cout (αντικείμενο εξόδου) είναι ένα αντικείμενο της κλάσης ostream (κλάση εξόδου)
- [Θα το μελετήσουμε σε επόμενο μάθημα]



A. Θεωρία

1. Κλάσεις

2. Ορισμός Κλάσης

- Ορίζουμε μία κλάση με την εξής δήλωση:

```
class class_name {  
  public:  
    // Δημόσιες μεταβλητές και συναρτήσεις  
  private:  
    // Ιδιωτικές μεταβλητές και συναρτήσεις  
};
```

- Μπορούμε να βάλουμε όποιο όνομα κλάσης θέλουμε (class_name)
- Με αυτήν τη δήλωση, κατασκευάσαμε έναν νέο τύπο δεδομένων
- Η δήλωση πρέπει να γίνεται πριν από τη main
- Ορίζουμε ένα αντικείμενο της κλάσης με τη δήλωση

```
class_name object_name;
```

- Ένα αντικείμενο είναι πλέον μια μεταβλητή του τύπου δεδομένων που έχει ορίσει η κλάση
 - και έχει αντίστοιχη συμπεριφορά με τις μεταβλητές (π.χ. μπορούμε να ορίσουμε πολλά αντικείμενα με τη δήλωση):

```
class_name object_name1, object_name2;
```



A. Θεωρία

1. Κλάσεις

3. Δημόσια (public) στοιχεία της κλάσης

- Τα δημόσια στοιχεία (μέλη – συναρτήσεις) ενός αντικειμένου της κλάσης είναι ορατά (προσβάσιμα) από οποιοδήποτε μέρος του προγράμματος.
 - Η πρόσβαση σε αυτά, εκτός του αντικειμένου της κλάσης (π.χ. από τη main), γίνεται με την τελεία (.), ως εξής:

`object.member`

- Ας δούμε ένα παράδειγμα:

```
class cow
{
    public:
        int weight;
    private:
        int hunger;
};
```

```
int main()
{
    cow molly;
    molly.weight = 500;
    // molly.hunger = 10; // Δεν επιτρέπεται. Είναι ιδιωτική μεταβλητή
}
```



A. Θεωρία

1. Κλάσεις

4. Ιδιωτικά (private) στοιχεία της κλάσης

- Τα ιδιωτικά στοιχεία (μέλη – συναρτήσεις) της κλάσης είναι ορατά (προσβάσιμα) ΜΟΝΟ από τις μεθόδους ενός αντικειμένου της κλάσης
 - Η πρόσβαση στα μέλη, γίνεται σαν να ήταν τοπικές μεταβλητές.
- Επεκτείνουμε το παράδειγμα μας:

```
class cow
{
    public:
        int weight;
        void express ()
        {
            if (hunger>10)
                cout<<"MmmmmmmmmMMMmmM";
        }
        int set_hunger (int h)
        {
            hunger = h;
        }
    private:
        int hunger;
};
```

```
int main()
{
    cow molly;
    molly.weight = 500;
    molly.set_hunger(49);
    molly.express();
}
```

Ολοκληρωμένο το πρόγραμμα είναι το:
CPP2.private.cpp

A. Θεωρία

1. Κλάσεις

5. Παράδειγμα (Προδιαγραφές)

- Θα κατασκευάσουμε μία κλάση με όνομα σκύλος.
- Ο σκύλος θα έχει μόνο ένα χαρακτηριστικό, την διάθεσή του, την οποία δεν μπορούμε να την ξέρουμε εκ των προτέρων.
 - άρα θα είναι μια ιδιωτική μεταβλητή.
 - Στην μοντελοποίηση, θα είναι μια ακέραια μεταβλητή η οποία,
 - αν είναι πάνω από 10, είναι σε καλή διάθεση,
 - αν είναι κάτω από 10, είναι σε κακή διάθεση
- Θα ζητάμε από το σκύλο να βγάζει ένα μήνυμα αν είναι σε καλή ή κακή διάθεση.
- Η υλοποίηση αυτών των προδιαγραφών (βλ. δεξιά)

```
class dog
{
    public:
        void report_mood()
        {
            if (mood>10)
                cout<<"I am cool";
            else
                cout<<"I am furious";
        }
    private:
        int mood;
};
```

Σημαντικό:

- Σχεδόν πάντα, τα δεδομένα της κλάσης (μεταβλητές) είναι ιδιωτικά μέλη.



A. Θεωρία

1. Κλάσεις

5. Παράδειγμα (Προδιαγραφές)

- Ο σκύλος θα έχει μία συνάρτηση, δημόσια, με την οποία θα γίνεται η αρχικοποίηση της μεταβλητής της διάθεσής του.
- Επίσης θα έχει μία δημόσια μέθοδο,
 - Με την οποία ο σκύλος θα γαβγίζει!

Παρατήρηση:

- Τις περισσότερες φορές, οι μεταβλητές - μέλη της κλάσης θα χρειάζονται αρχικοποίηση
- Γι' αυτό η C++, ορίζει έναν ειδικό τρόπο αρχικοποίησης, με τις συναρτήσεις constructor (κατασκευαστές)
 - τις οποίες θα μελετήσουμε στη συνέχεια του μαθήματος

```
class dog
{
    public:
        void init(int in_mood)
        {
            mood=in_mood;
        }
        void bark()
        {
            cout<<"woof"<<endl;
        }
        void report_mood()
        {
            if (mood>10)
                cout<<"I am cool";
            else
                cout<<"I am furious";
        }
    private:
        int mood;
};
```




A. Θεωρία

1. Κλάσεις

5. Παράδειγμα (Προδιαγραφές)

- Η main θα αναλάβει το ρόλο του συντονιστή των αντικειμένων.
- Συγκεκριμένα θα υλοποιήσει έναν διάλογο μεταξύ του σκύλου του και του αφεντικού του, ο οποίος θα είναι ο ακόλουθος:

Διάλογος

- Ιδιοκτήτης Πίκο: Πίκο πως είσαι;
 - Πίκο: Έχω τα νεύρα μου
 - Ιδιοκτήτης: Γάυγισε δύο φορές σε παρακαλώ
 - Πίκο: Γαβ, γαβ!
- Σημαντική είναι και η αρχικοποίηση της μεταβλητής μέλους της διάθεσης
 - (Για να έχει τα νεύρα του, θα πρέπει να έχει τιμή < 10)

```
int main()
{
    dog piko;

    piko.init(6);

    cout<<"Piko, how are you today?"<<endl;
    piko.report_mood();
    cout<<endl;

    cout<<"Piko, please bark twice for
        me"<<endl;
    piko.bark();
    piko.bark();

    return 0;
}
```



A. Θεωρία

1. Κλάσεις

5. Παράδειγμα (Προδιαγραφές)

Παρατηρήσεις:

- Η μοντελοποίηση των προδιαγραφών στο σκεπτικό των κλάσεων, είναι η πιο δύσκολη και σημαντική δραστηριότητα στη C++
 - Απαιτεί μεγάλη εμπειρία και εφάπτεται στον τομέα της Ανάλυσης Συστημάτων.
- Σημαντικός και ο ρόλος της `main` ως ο συνδετικός κρίκος μεταξύ των αντικειμένων
 - Είναι συχνά ο μεσάζοντας της επικοινωνίας
 - Αναλαμβάνει το ρόλο:
 - να προετοιμάσει τα αντικείμενα (initialization)
 - να ενεργοποιήσει την επικοινωνία τους
- Είναι σαν τον σεναριογράφο, ο οποίος χρησιμοποιεί τους ήρωές του (αντικείμενα), για να υλοποιήσει μία ιστορία.



A. Θεωρία

2. Περισσότερα για τις Κλάσεις

1. Ορισμός συναρτήσεων έξω από την κλάση

- **Συνηθίζεται οι μέθοδοι να ορίζονται εκτός της κλάσης**, για δύο λόγους:
 - Είναι «καλύτερο» να έχουμε μόνο τα μέλη (μεταβλητές) και τα πρωτότυπα συναρτήσεων (μέθοδοι) της κλάσης
 - ώστε να έχουμε μια γρήγορη αναφορά, χωρίς να χρειάζεται να σκρολάρουμε σε πολλές γραμμές κώδικα.
 - για να χωρίζουμε (σε μεγάλες κλάσεις) τη δήλωση από την υλοποίηση σε ξεχωριστά αρχεία
 - (θα το δούμε στη συνέχεια του μαθήματος)
- Για να ορίσουμε το σώμα της μεθόδου εκτός της κλάσης:
 - Κρατάμε **μόνο το πρωτότυπο μέσα στη κλάση**
 - Εκτός της κλάσης (μετά τη main) ορίζουμε το σώμα της συνάρτησης:
 - **όπου θα πρέπει να προηγείται μπροστά από το όνομα της συνάρτησης το όνομα της κλάσης, ακολουθούμενο από τον τελεστή ::** (τελεστής επίλυσης εμβέλειας):

```
return_type class_name::method_name(arguments)
```

- Ο **τελεστής επίλυσης εμβέλειας** (resolution scope operator) :: έχει πολλές χρήσεις κι έχουμε δει ήδη δύο:
 - χώρος_ονομάτων::αντικείμενο (π.χ. std::cout)
 - κλάση::συνάρτηση (π.χ. dog::bark())



A. Θεωρία

2. Περισσότερα για τις Κλάσεις

1. Ορισμός συναρτήσεων έξω από την κλάση

- Ας δούμε την υλοποίηση με τα σώματα συναρτήσεων έξω από την κλάση:

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

class dog
{
    public:
        void init(int in_mood);
        void bark();
        void report_mood();
    private:
        int mood;
};

int main()
{
    ....
}
```

```
void dog::init(int in_mood)
{
    mood=in_mood;
}

void dog::bark()
{
    cout<<"woof!"<<endl;
}

void dog::report_mood()
{
    if (mood>10)
        cout<<"I am cool";
    else
        cout<<"I am furious";
}
```



A. Θεωρία

2. Περισσότερα για τις Κλάσεις

2. Παρουσίαση Ιδιωτικών – Δημόσιων Μέλων μιας κλάσης

- Είδαμε ότι ο τρόπος που παρουσιάζουμε τα μέλη και τις μεθόδους
 - Είναι γράφοντας πρώτα τα δημόσια μέλη-μεθόδους και έπειτα τα ιδιωτικά μέλη-μεθόδους

```
class X {  
    public:  
        ...  
    private:  
        ...  
};
```

- Ωστόσο αυτό είναι «μεθοδολογικό», συνηθίζεται από αρκετούς προγραμματιστές.
- Θα μπορούσαμε να εναλλάσσουμε τα μέλη από ιδιωτικά σε δημόσια, αναλόγως με τις ορέξεις μας:

```
class X {  
    public:  
        ...  
    private:  
        ...  
    public:  
        ....  
};
```

- Ωστόσο αυτό δεν συνηθίζεται και είναι αντιαισθητικό.



A. Θεωρία

2. Περισσότερα για τις Κλάσεις

2. Παρουσίαση Ιδιωτικών – Δημόσιων Μέλων μιας κλάσης

- Ένας συνηθισμένος τρόπος γραφής των κλάσεων είναι και ο ακόλουθος:

```
class X {  
    ...  <- Ότι γράψουμε εδώ είναι private  
    public:  
    ...  
};
```

- Εδώ τα αρχικά μέλη-μέθοδοι που γράφουμε αμέσως μετά το άγκιστρο, χωρίς να έχουμε ορίσει την πρόσβαση, ορίζονται από τη γλώσσα να είναι ιδιωτικά.

- Στα πλαίσια αυτών των σημειώσεων, έχει επιλεγεί ο 1^{ος} τρόπος που αναφέραμε.



A. Θεωρία

2. Περισσότερα για τις Κλάσεις

3. Χωρισμός σε αρχεία

- Η συνηθισμένη πολιτική για μεγάλα projects είναι η εξής:
 - Η δήλωση της κλάσης γίνεται σε ένα αρχείο με προέκταση .h (π.χ. class.h)
 - Τα σώματα των συναρτήσεων γίνονται σε ένα αρχείο με προέκταση .cpp (π.χ. class.cpp)
 - Το αρχείο αυτό κάνει #include τη δήλωση της κλάσης με διπλά εισαγωγικά:
 - Το αρχείο αυτό μπορεί να μεταγλωττιστεί, ώστε να παραχθεί ένα ενδιάμεσο αρχείο (π.χ. class.o) (αντικειμενικό αρχείο)
- Η main βρίσκεται σε ξεχωριστό αρχείο, το οποίο περιέχει την ενσωμάτωση της «βιβλιοθήκης» της κλάσης που έχουμε κατασκευάσει.

```
#include "class.h"
```

- Σχηματικά (για δύο κλάσεις):

class1.h

Ορισμός της
κλάσης 1

class2.h

Ορισμός της
κλάσης 2

class1.cpp

```
#include "class1.h"
```

Σώματα των μεθόδων

class2.cpp

```
#include "class2.h"
```

Σώματα των μεθόδων

main.cpp

```
#include "class1.h"  
#include "class2.h"
```

Συνάρτηση main



A. Θεωρία

2. Περισσότερα για τις Κλάσεις

3. Χωρισμός σε αρχεία

- Ας δούμε τον χωρισμό σε αρχεία στο παράδειγμα μας:

```
/* dog.h */
```

```
class dog
```

```
{
```

```
    public:
```

```
        void init(int in_mood);
```

```
        void bark();
```

```
        void report_mood();
```

```
    private:
```

```
        int mood;
```

```
};
```

```
/* dog.cpp */
```

```
#include <iostream>
```

```
#include "dog.h"
```

```
using namespace std;
```

```
void dog::init(int in_mood)
```

```
{
```

```
    mood=in_mood;
```

```
}
```

```
void dog::bark()
```

```
{
```

```
    cout<<"woof!"<<endl;
```

```
}
```

```
void dog::report_mood()
```

```
{
```

```
    if (mood>10)
```

```
        cout<<"I am cool";
```

```
    else
```

```
        cout<<"I am furious";
```

```
}
```

```
/* main.cpp */
```

```
#include <iostream>
```

```
#include "dog.h"
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    dog piko;
```

```
    piko.init(6);
```

```
    cout<<"Piko, how are you  
            today?"<<endl;
```

```
    piko.report_mood();
```

```
    cout<<endl;
```

```
    cout<<"Piko, please bark  
            twice for me"<<endl;
```

```
    piko.bark();
```

```
    piko.bark();
```

```
    return 0;
```

```
}
```




A. Θεωρία

2. Περισσότερα για τις Κλάσεις

3. Χωρισμός σε αρχεία (onlineGDB)

- Βλέπουμε και πως γίνεται η υλοποίηση στο περιβάλλον του onlineGDB
- Δημιουργούμε ένα καινούριο project και έπειτα με το κουμπί νέου αρχείου



- Δημιουργούμε τα τρία αρχεία (και τους θέτουμε και τα αντίστοιχα ονόματα)
- Με το κουμπί “Run” το onlineGDB κάνει όλες τις απαραίτητες ενέργειες:
 - Προεπεξεργασία (preprocessing):
 - Οι εντολές `#include` αντικαθίστανται από τον κώδικα των αρχείων στα οποία αναφέρονται (και αντίστοιχα για άλλες οδηγίες του προεπεξεργαστή)
 - Μεταγλώττιση (compiling):
 - Κάθε αρχείο `.cpp` μεταγλωττίζεται μόνο του και παράγει ένα νέο αρχείο (συνήθως με προέκταση `.o`) το οποίο έχει τη μετάφραση του αρχείου σε γλώσσα μηχανής και λέγεται αντικειμενικό αρχείο
 - Σύνδεση (linking):
 - Τα αντικειμενικά αρχεία (`.o`) συνδυάζονται σε ένα τελικό εκτελέσιμο αρχείο (`.exe`)
- Στο onlineGDB δεν βλέπουμε τα ενδιάμεσα στάδια και τρέχει αμέσως το τελικό εκτελέσιμο.



A. Θεωρία

3. Ειδικές Μέθοδοι Κλάσεων

1. Γενικά

Σχεδόν σε κάθε κλάση θα ορίζουμε 3 οικογένειες μεθόδων που κάνουν συνηθισμένες εργασίες

- Ο **κατασκευαστής** (constructor)
 - Έχει το ρόλο του να αρχικοποιήσει τα μέλη της κλάσης.
 - Έχει ειδικό συντακτικό, το οποίο ορίζεται από τη C++
 - Καλείται αυτόματα.
- Ο **καταστροφέας** (destructor)
 - Κάνει ενέργειες όταν ένα αντικείμενο καταστρέφεται
 - Έχει ειδικό συντακτικό, το οποίο ορίζεται από τη C++
 - Καλείται αυτόματα.
- Οι **ελεγκτές πρόσβασης** (accessors)
 - Ελέγχουν την πρόσβαση στα δεδομένα – μέλη της κλάσης
 - Καλούνται από τον προγραμματιστή.

- Παρατήρηση: Υπάρχει και ο constructor αντιγράφου (copy constructor)
 - που επίσης καλείται αυτόματα και έχει ειδικό συντακτικό.
 - και θα τον μελετήσουμε σε επόμενο μάθημα.



A. Θεωρία

3. Ειδικές Μέθοδοι Κλάσεων

2. Κατασκευαστής (constructor)

- Ο **κατασκευαστής (constructor)** είναι δημόσια μέθοδος μιας κλάσης, η οποία:
 - καλείται αυτόματα όταν κατασκευάζεται ένα αντικείμενο της κλάσης.
- Η δήλωση του κατασκευαστή γίνεται στο δημόσιο μέρος της κλάσης.
- Προσοχή:
 - Έχει **υποχρεωτικά ίδιο όνομα με την κλάση** (με αυτόν τον τρόπο ορίζεται ότι είναι κατασκευαστής) και **δεν έχει τύπο επιστροφής** (δεν επιστρέφει τίποτα)
 - Μπορούμε να έχουμε **πολλούς κατασκευαστές** στην ίδια κλάση με υπερφόρτωσή του (constructor overloading)
 - **Ο ρόλος του είναι να αρχικοποιήσει** όσα μέλη (μεταβλητές) απαιτείται από την σχεδιάσή μας.
- Η δήλωση του πρωτοτύπου θα γίνεται ως:

```
class_name(arguments);
```

- Η δήλωση του σώματος θα γίνεται ως:

```
class_name::class_name(arguments);
```

- Ο constructor πιθανότατα θα υπάρχει σε κάθε κλάση που θα φτιάξουμε στα επόμενα μαθήματα



A. Θεωρία

3. Ειδικές Μέθοδοι Κλάσεων

2. Κατασκευαστής (constructor)

- Διορθώνουμε τον ορισμό της κλάσης “σκύλος” ώστε να έχει constructor.

```
/* constructor.cpp */  
  
#include <iostream>  
using namespace std;  
  
class dog  
{  
public:  
    dog(int in_mood);  
    void bark();  
    void report_mood();  
private:  
    int mood;  
};
```

```
dog::dog(int in_mood)  
{  
    mood=in_mood;  
}  
void dog::bark()  
{  
    cout<<"woof!"<<endl;  
}  
void dog::report_mood()  
{  
    if (mood>10)  
        cout<<"I am cool";  
    else  
        cout<<"I am furious";  
}
```

```
int main()  
{  
    dog piko(6);  
    cout<<"Piko, how are you  
        today?"<<endl;  
    piko.report_mood();  
    cout<<endl;  
    cout<<"Piko, please bark  
        twice for me"<<endl;  
    piko.bark();  
    piko.bark();  
    return 0;  
}
```

- Παρατηρούμε ότι κατά τη δήλωση του αντικειμένου, βάζουμε τα ορίσματα του constructor.
- Μπορούμε να ορίσουμε και constructor χωρίς ορίσματα,
 - Θα καλείται κατά τη δήλωση του αντικειμένου (δεν απαιτεί παρενθέσεις)



A. Θεωρία

3. Ειδικές Μέθοδοι Κλάσεων

3. Καταστροφέας (destructor)

- Ο **καταστροφέας (destructor)** είναι δημόσια μέθοδος μιας κλάσης, η οποία:
 - καλείται αυτόματα όταν καταστρέφεται ένα αντικείμενο της κλάσης.
- Η δήλωση του καταστροφέα γίνεται στο δημόσιο μέρος της κλάσης.
- Προσοχή:
 - Έχει **υποχρεωτικά ίδιο όνομα με την κλάση που μπροστά έχει περισπωμένη ~** (με αυτόν τον τρόπο ορίζεται ότι είναι καταστροφέας), **δεν έχει τύπο επιστροφής** (δεν επιστρέφει τίποτα) και **δεν παίρνει ορίσματα**.
 - Κάθε κλάση μπορεί να έχει **το πολύ έναν καταστροφέα**
 - **Ο ρόλος του είναι να μεριμνήσει** για ενέργειες που πρέπει να γίνουν πριν το αντικείμενο καταστραφεί
 - Η συχνότερη ενέργεια είναι να απελευθερώσει μνήμη που έχουμε δεσμεύσει δυναμικά (θα το δούμε σε επόμενο μάθημα)
- Η δήλωση του πρωτοτύπου θα γίνεται ως:

```
~class_name();
```

- Η δήλωση του σώματος θα γίνεται ως:

```
class_name::~~class_name();
```



A. Θεωρία

3. Ειδικές Μέθοδοι Κλάσεων

3. Καταστροφέας (destructor)

- Προσθέτουμε και έναν (διακοσμητικό) καταστροφέα στην κλάση μας

```
/* destructor.cpp */

#include <iostream>
using namespace std;

class dog
{
public:
    dog(int in_mood);
    ~dog();
    void bark();
    void report_mood();
private:
    int mood;
};
```

```
dog::dog(int in_mood)
{
    mood=in_mood;
}

dog::~~dog()
{
    cout<<"I will be waiting...!";
}

void dog::bark()
{
    cout<<"woof!"<<endl;
}

void dog::report_mood()
{
    if (mood>10)
        cout<<"I am cool";
    else
        cout<<"I am furious";
}
```

```
int main()
{
    dog piko(6);
    cout<<"Piko, how are you
           today?"<<endl;
    piko.report_mood();
    cout<<endl;

    cout<<"Piko, please bark
           twice for me"<<endl;
    piko.bark();
    piko.bark();
    return 0;
}
```



A. Θεωρία

3. Ειδικές Μέθοδοι Κλάσεων

3. Καταστροφέας (destructor)

- Μία τοπική μεταβλητή (όπως το αντικείμενο που έχουμε στο πρόγραμμά μας):
 - Δεσμεύει χώρο μνήμης κατά τη δήλωση της
 - Αποδεσμεύει τον χώρο μνήμης της, όταν ολοκληρώνεται η συνάρτηση στην οποία έχει δηλωθεί.
- Έτσι ο destructor, στο παράδειγμα μας, θα κληθεί όταν ολοκληρώνεται η συνάρτηση main.

```
Piko, how are you today?  
I am furious  
Piko, please bark twice for me  
woof!  
woof!  
I will be waiting...!
```



A. Θεωρία

3. Ειδικές Μέθοδοι Κλάσεων

4. Accessors (setters – getters)

- Οι **accessors** (ελεγκτές πρόσβασης στα μέλη της κλάσης):
 - είναι μια **προγραμματιστική πρακτική**, που λέει ότι όλα τα μέλη (μεταβλητές) της κλάσης πρέπει να είναι ιδιωτικά.
 - Αν θέλουμε να αλλάζουμε την τιμή ενός μέλους, θα ορίζουμε μία συνάρτηση που:
 - θα είναι δημόσια,
 - θα παίρνει ως όρισμα τη νέα τιμή, και θα αλλάζει την τιμή του μέλους.
 - θα έχει την εξής μορφή (αν το μέλος π.χ. ονομάζεται x και είναι ακέραιος):

```
void set_x (int new_x) {  
    x = new_x;  
}
```

- Η συνάρτηση αυτή θα λέγεται **setter**.
- Αν θέλουμε να παίρνουμε την τιμή ενός μέλους, θα ορίζουμε μία συνάρτηση που:
 - θα είναι δημόσια και θα επιστρέφει την τιμή.

```
int get_x () const {  
    return x;  
}
```

- Η συνάρτηση αυτή θα λέγεται **getter**.



A. Θεωρία

3. Ειδικές Μέθοδοι Κλάσεων

4. Accessors (setters – getters)

- Επεκτείνουμε την κλάση «σκύλος» με τους accessors της μεταβλητής mood:

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

class dog
{
public:
    dog(int in_mood);
    ~dog();
    void set_mood(int in_mood);
    int get_mood() const;
    void bark();
    void report_mood();
private:
    int mood;
};
```

```
int main() {
    ...
}

...
void dog::set_mood(int in_mood)
{
    mood = in_mood;
}
int dog::get_mood() const
{
    return mood;
}
...
```



A. Θεωρία

3. Ειδικές Μέθοδοι Κλάσεων

4. Accessors (setters – getters)

- Οι accessors θεωρούνται απαραίτητη προγραμματιστική πρακτική, διότι:
 - Δεν επιτρέπεται στον πελάτη της κλάσης να έχει πρόσβαση στα πραγματικά δεδομένα.
 - Συνεπώς αυξάνει την προληπτικότητα απέναντι σε κάποια λάθος χρήση.
 - Δίνει τη δυνατότητα στον προγραμματιστή της κλάσης,
 - να αλλάξει την υλοποίηση της κλάσης,
 - χωρίς να επηρεαστεί ο κώδικας που ήδη έχει γραφεί και χρησιμοποιεί την κλάση.



A. Θεωρία

3. Ειδικές Μέθοδοι Κλάσεων

4. Accessors (setters – getters) (1. Μέθοδοι const)

- Παρατηρήστε ότι ο getter της κλάσης έχει τη λέξη κλειδί **const** να ακολουθεί τη δήλωσή της
 - Με τον τρόπο αυτό ορίζουμε ότι **η μέθοδος δεν έχει δικαίωμα να επηρεάσει τα μέλη της κλάσης.**
 - Θεωρείται ότι είναι καλή προγραμματιστική πρακτική να ορίζουμε μία μέθοδο ως const
 - όταν γνωρίζουμε ότι δεν πρέπει να επηρεάσει τα μέλη της κλάσης,
 - σαν μία ακόμη ασφαλιστική δικλείδα απέναντι σε προγραμματιστικά λάθη που μπορεί να προκύψουν.
 - Η λέξη-κλειδί const:
 - πρέπει να ακολουθεί τόσο τη δήλωση του πρωτοτύπου της μεθόδου
 - όσο και τη δήλωση του σώματος της μεθόδου



B. Ασκήσεις

Άσκηση 1: Κλάση με πολλαπλούς constructors

Κατασκευάστε ένα πρόγραμμα, το οποίο να υλοποιεί μια κλάση με όνομα dummy η οποία:

- Να έχει μια ακέραια μεταβλητή ως ιδιωτικό μέλος με όνομα x.
- Να έχει δύο κατασκευαστές:
 - έναν χωρίς ορίσματα, που να αρχικοποιεί τη μεταβλητή x σε 0.
 - έναν με όρισμα την τιμή που θα πάρει η μεταβλητή x
- Να έχει accessors
- Να έχει destructor
 - Με ένα απλό ενημερωτικό μήνυμα, ότι καταστρέφεται το αντικείμενο με την τιμή που έχει η μεταβλητή x.

Και μία συνάρτηση main η οποία

- Να δηλώνει δύο αντικείμενα dummy
 - Το ένα να αρχικοποιεί την μεταβλητή x σε 10 (μέσω του constructor)
 - Το άλλο να μην αρχικοποιεί τη μεταβλητή x



B. Ασκήσεις

Άσκηση 2: Κλάση Σημείο

Ορίστε μία κλάση που να απεικονίζει ένα σημείο του επιπέδου

- Θα έχει ως μέλη τις συντεταγμένες του x , y (πραγματικές)
- Θα έχει constructor χωρίς ορίσματα που θα αρχικοποιεί το σημείο στο $(0,0)$
- Θα έχει constructor με ορίσματα τις συντεταγμένες του νέου σημείου
- Θα έχει accessors
- Θα έχει μια δημόσια συνάρτηση εκτύπωσης που θα τυπώνει τις συντεταγμένες στη μορφή (x,y)

Ορίστε και μία συνάρτηση `main`

- Η οποία θα δημιουργεί 3 σημεία της αρεσκείας σας.
- Θα τα τυπώνει στην οθόνη.



B. Ασκήσεις

Άσκηση 3: Μία κλάση για αποθήκευση ατόμου

Ο Μάγος Gandalf

- έχει ηλικία 2019 ετών (να γίνεται η αρχικοποίηση μέσω constructor)
- το μούσι του είναι χρώμα γκρι (συμβολοσειρά, να αρχικοποιείται μέσω constructor)
- έχει έναν δείκτη μαγείας (να αρχικοποιείται στο 100 μέσω constructor και να έχει accessors)
- Μπορεί να κάνει τα εξής ξόρκια:
 - Fireball Spell (κοστίζει 30 πόντους μαγείας)
 - Lightning Spell (κοστίζει 90 πόντους μαγείας)
- Μπορεί να περιμένει (wait), προσθέτοντας 10 πόντους μαγείας στον δείκτη του

Υλοποιήστε ένα σενάριο στο οποίο ο Gandalf

- Κάνει ένα fireball spell
- Περιμένει
- Κάνει ένα lightning spell
- Περιμένει
- Περιμένει
- Κάνει ένα fireball spell

Σε κάθε βήμα του σεναρίου να τυπώνεται στην οθόνη

- Ένα μήνυμα με το τι κάνει ο Gandalf
- Η τιμή του μετρητή μαγείας του