Classes - objects

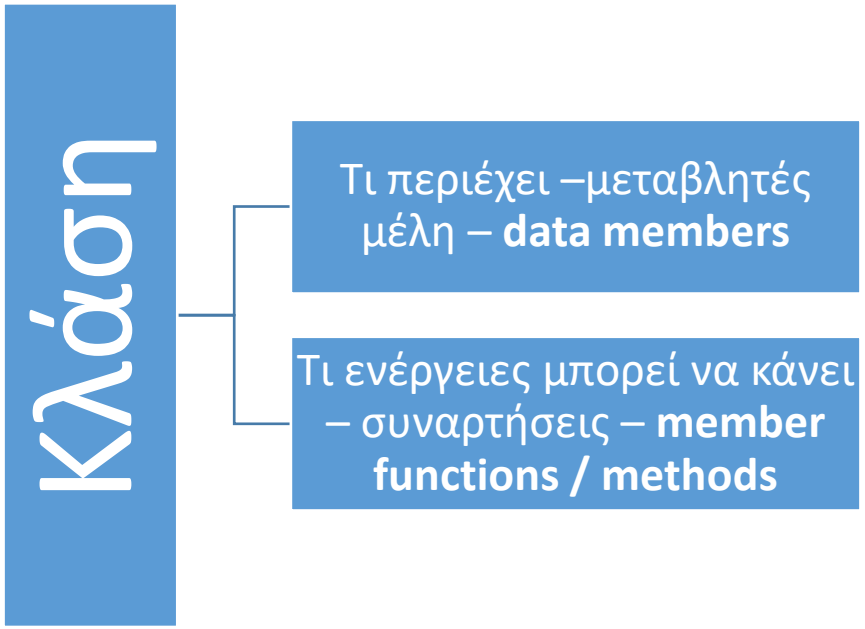
Classes

Κλάση (class)

- Οι αντικειμενοστραφείς (object oriented) γλώσσες προγραμματισμού δίνουν την δυνατότητα στον προγραμματιστή να ορίσει τους δικούς του τύπους δεδομένων
- Με αυτόν τον τρόπο μπορεί να περιγράψει τα «αντικείμενα» με τα οποία ασχολείται το πρόγραμμα που δημιουργεί
  - Π.χ. τα σημεία του χώρου για ένα σχεδιαστικό πρόγραμμα
- Οι καινούργιοι τύποι που δημιουργεί ο προγραμματιστής ονομάζονται ***κλάσεις (classes)***

Περιγραφή μίας κλάσης

- Δημιουργία μιας κλάσης καινούργιου τύπου
- Για παράδειγμα ένα σημείο περιγράφεται
  - Από τις συντεταγμένες x, y
  - Από τις ενέργειες που μπορούν να γίνουν όπως πχ μετακίνηση κατά τον χ, κατά τον y, συμμετρικό κ.α.



Δήλωση μιας κλάσης

- Η δήλωση μιας κλάσης έχει την εξής μορφή
  - Το όνομα της κλάσης έχει το πρώτο χαρακτήρα κεφαλαίο (προαιρετικό)
  - Ο κώδικας κάθε κλάσης γράφεται σε ξεχωριστό αρχείο (προαιρετικό)
  - Δύο απλές μεταβλητές και μία μέθοδος (επιστρέφει ακέραιο 1,2,3,4 ανάλογα με το ποιο τεταρτημόριο βρίσκεται το σημείο)

```
class Point
{
public:
    double x;
    double y;
    int quartile();
};
```

Συνάρτηση – μέλος (μέθοδος)

Class scope

- Κάθε μία κλάση ορίζει την δική της εμβέλεια (scope) ονομάτων.
- Μάλιστα στις κλάσεις υπάρχει και το εξής παράδοξο μπορούμε να χρησιμοποιήσουμε ένα όνομα πριν το βρούμε.
  - Αυτό γίνεται γιατί ο compiler πρώτα ελέγχει τα ονόματα και μετά κάνει την αποτίμηση τους

```
class Point
{
public:
    double x=y;
    double y;

    int quartile() {
        return x * y;
    }
};
```

Το y δεν έχει δηλωθεί ακόμα!

Μέθοδοι έχουν πρόσβαση στα μέλη

Το παράδειγμα έχει σκοπό να παρουσιάσει την λειτουργία του class scope

Ορισμός συνάρτησης μέλους

- Ο ορισμός μιας μεθόδου μπορεί να γίνει και εκτός κλάσης.
- Στην επικεφαλίδα του ορισμού αναφέρεται και σε ποια κλάση ανήκει αυτή η μέθοδος (class scope)
- **Ο κώδικας στις μεθόδους ΕΧΕΙ ΠΡΟΣΒΑΣΗ στις μεταβλητές μέλη**

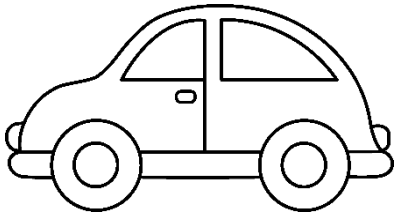
```
int Point::quartile() {
    if (x >= 0) {
        if (y >= 0)
            return 1;
        else
            return 2;
    }
    else {
        if (y >= 0)
            return 4;
        else
            return 3;
    }
}
```

Ο τελεστής ::

- Η χρήση του τελεστή :: ονομάζεται ***scope resolution*** και σκοπός του είναι να διευκρινίζει σε ποιο scope ανήκει ένα όνομα που χρησιμοποιούμε
- Στην συγκεκριμένη περίπτωση αναφέρει ότι η συγκεκριμένη μέθοδος ανήκει στην αντίστοιχη κλάση
- Θα δούμε χρήσεις του και στην κληρονομικότητα καθώς και σε στατικά μέλη κλάσεων

Αντικείμενο - object

- ***Αντικείμενο – object*** είναι μία μεταβλητή που ο τύπος της δεν είναι απλός αλλά είναι κλάση
- Για παράδειγμα μια μεταβλητή που περιγράφει ένα συγκεκριμένο σημείο στον χώρο είναι αντικείμενο της κλάσης Point
  - Υπάρχει «υπερφόρτωση» του όρου αντικείμενο. Πολλές φορές χρησιμοποιείται και για να περιγράψει κάτι το οποίο υπάρχει στην μνήμη



class

objects

Δημιουργώντας objects

- Η δημιουργία ενός object γίνεται με τον ίδιο τρόπο με την δημιουργία μιας μεταβλητής απλού τύπου
- Μπορούμε μάλιστα να δημιουργήσουμε ακόμα και δυναμικά αντικείμενα με την χρήση του new
- Αν έχουμε γράψει τον κώδικα της class σε άλλο αρχείο θα πρέπει να το κάνουμε include

```
#include "pch.h"
#include"Point.h"
#include <iostream>

int main()
{
    Point a;
    Point *p;

    p = new Point;


}
```

Πρόσβαση στα πεδία ενός object

- Η πρόσβαση στις μεταβλητές μέλη (πεδία – fields) ενός αντικειμένου γίνεται
- Με τον τελεστή (.) αν χρησιμοποιούμε το ίδιο το αντικείμενο
- Με τον τελεστή (->) αν χρησιμοποιούμε pointer στο αντικείμενο
- Οι μεταβλητές μέλη χρησιμοποιούνται σαν τις κανονικές μεταβλητές

```
Point a;
Point *p;

p = new Point;

a.x = 1;
a.y = 3;
p->x = -2;
p->y = 4;
```

Χρήση των μεθόδων

- Στις συναρτήσεις μέλη (member functions – methods) η πρόσβαση γίνεται με τον ίδιο τρόπο με τα πεδία
- Πρέπει να χρησιμοποιούνται οι παρενθέσεις και να δίνονται τα απαραίτητα ορίσματα
- Επίσης μπορεί να επιστρέφουν και τιμή

```
Point a;
Point *p;

p = new Point;

a.x = 1;
a.y = 3;
p->x = -2;
p->y = 4;

cout << a.quartile() << endl;
cout << p->quartile() << endl;
```

Encapsulation - Abstraction

Πρόβλημα…

- Αυτή την στιγμή ο κώδικας μας αποτελείται
  - Από την class Point που ορίζει το τι περιέχουν και πως συμπεριφέρονται τα αρχεία
  - Από την συνάρτηση main που χρησιμοποιεί την class Point μέσω των αντικειμένων
- Αν αποφασίσουμε να αλλάξουμε τον τρόπο περιγραφής των Points (π.χ. δύο ακεραίους (int,int) για κάθε σημείο -> ***Πρέπει να αλλάξουμε και την main***
- ***Θέλουμε να μπορούμε να αλλάζουμε την εσωτερική περιγραφή των κλάσεων χωρίς να επηρεάζεται ο κώδικας που ήδη τις χρησιμοποιεί!***

Encapsulation – abstraction

- Encapsulation
  - Είναι η τεχνική με την οποία ενσωματώνουμε τα δεδομένα/πληροφορίες (μεταβλητές – μέλη) μαζί με τον κώδικα που τις επεξεργάζεται (μέθοδοι) μέσα σε ένα δοχείο (class – object)
- Abstraction
  - Είναι η τεχνική με την οποία «αποκρύπτουμε» από τον χρήστη την δομή με την οποία έχει αναπαρασταθεί ένα αντικείμενο. Το χρήστη τον ενδιαφέρει πως θα την χρησιμοποιήσει όχι πως είναι αποθηκευμένη εσωτερικά

Access specifier - private

- Ο τρόπος με τον οποίο μπορούμε να υποστηρίξουμε την τεχνική του abstraction στις classes είναι με την χρήση των access specifiers
- Με την χρήση του ***private*** access specifier καθορίζουμε ότι στα συγκεκριμένα μέλη της κλάσης μπορούν να έχει πρόσβαση μόνο κώδικας που ανήκει στην κλάση (και ΟΧΙ η main)

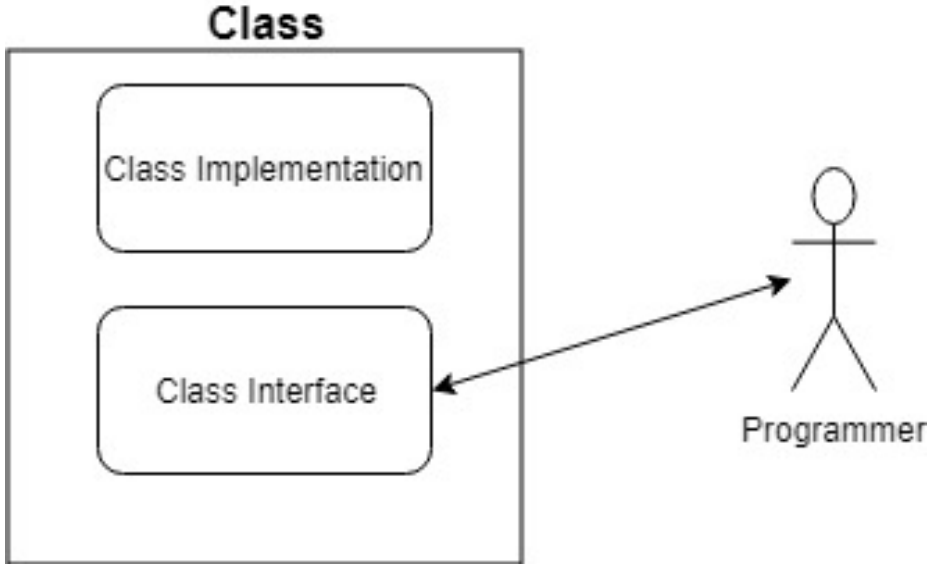
```
class Point
{
private:
    double x;
    double y;
public:
    int quartile();
};
```

Πρόσβαση μόνο από κώδικα που ανήκει στην κλάση (π.χ. μέθοδοι)

Πρόσβαση από οποιοδήποτε κώδικα

Class design

- Σχεδιάζουμε τις κλάσεις έτσι ώστε ο προγραμματιστής να έχει πρόσβαση σε αυτές μέσω ενός υποσυνόλου μεθόδων που ονομάζεται ***interface***
- Ο προγραμματιστής δεν χρειάζεται να γνωρίζει την εσωτερική αναπαράσταση της κλάσης (design) για να την χρησιμοποιήσει
- ***Μπορούμε να μεταβάλουμε το design όσο αυτό ΔΕΝ επηρεάζει το interface***



Πρόσβαση στα private members

- Για να μπορούμε να έχουμε πρόσβαση στα private members δημιουργούμε public μεθόδους που αυτές με την σειρά τους έχουν πρόσβαση στα private members
  - ***Accessor*** είναι η μέθοδος η οποία επιστρέφει την τιμή της private member variable
  - ***Mutator*** είναι η μέθοδος που μεταβάλει την τιμή μιας private member variable
- Η δημιουργία τους εξαρτάται από την προβλεπόμενη χρήση της κλάσης (δεν είναι απαραίτητη)
- Η μεταβολή, και η επιστροφή, των τιμών των private member variables είναι πλέον ελεγχόμενη από «δικό μας» κώδικα!

Χρήση accessor - mutator

```
class Point
{
private:
    double x;
    double y;
public:
    int quartile();
    void set_x(double a);
    void set_y(double b);
    double get_x();
    double get_y();
};
```

```
void Point::set_y(double b)
{
    y = b;
}

double Point::get_x()
{
    return x;
}
```

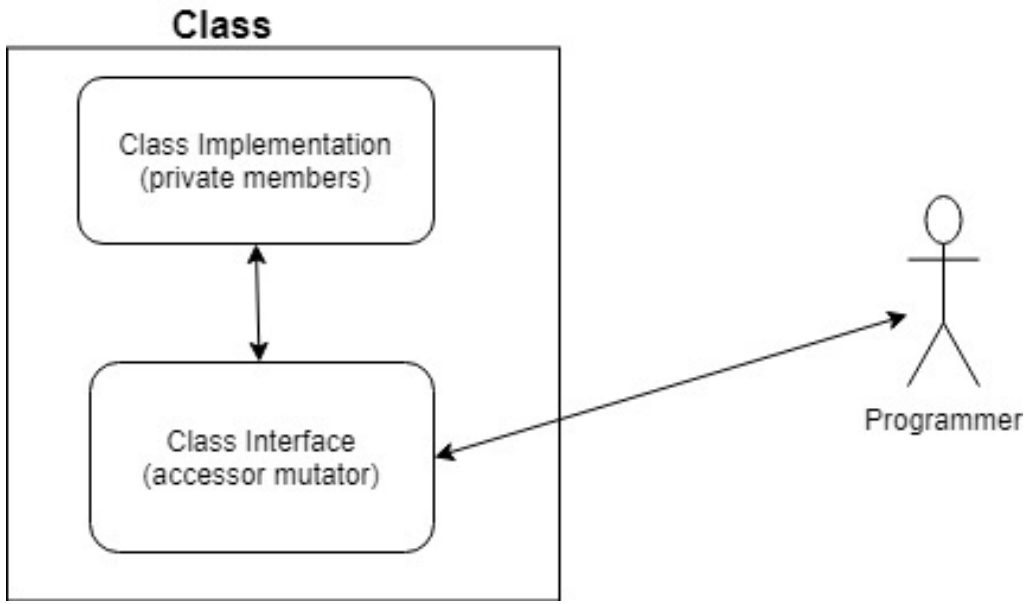
Χρήση accessor - mutator

```
Point a;
Point *p;

p = new Point;

a.set_x(1);
a.set_y(3);
p->set_x(-2);
p->set_y(4);

cout << a.quartile() << endl;
cout << p->quartile() << endl;
```



Τελεστής ανάθεσης

- Ο τελεστής ανάθεσης ισχύει για όλες τις νέες κλάσεις που δημιουργούμε
- Μπορούμε να αναθέσουμε σε ένα αντικείμενο που δημιουργούμε την τιμή ενός άλλου αντικειμένου (της ίδιας κλάσης)
- Αποτέλεσμα είναι να πάρουν οι μεταβλητές μέλη τις αντίστοιχες τιμές.

```
p = new Point;

a.set_x(1);
a.set_y(3);
p->set_x(-2);
p->set_y(4);

b = a;
// b.x=1 kai b.y=3
cout << b.quartile() << endl
```

Constructors / destructors

Constructors

- Για την αρχικοποίηση των objects υπάρχει ένας μηχανισμός που ονομάζεται **constructor**
- Ο constructor είναι μια εξειδικευμένη μέθοδος της κλάσης
- Σκοπός του constructor είναι να αρχικοποιεί τα data members
- Ο constructor χρησιμοποιείται κάθε φορά που δημιουργείται ένα αντικείμενο (είτε απλό είτε δυναμικό)
- Σε περίπτωση που δεν δημιουργήσουμε constructor ο compiler δημιουργεί μόνος του έναν ***synthesized default constructor***

Δημιουργία constructor

```
class Point
{
private:
    double x;
    double y;
public:
    Point();
    Point(double a, double b);
```

Μέθοδος της κλάσης που

- Έχει το **ΙΔΙΟ ΟΝΟΜΑ** με την κλάση
- **ΔΕΝ** επιστρέφει τιμή (ούτε void)
- Είναι στο public section

- Μπορούμε να υπερφορτώσουμε τον constructor
  - Ο constructor που δεν έχει παραμέτρους λέγεται **default**

Προσοχή στο συντακτικό! Constructor initializer list

Μπορεί να έχει και άλλο κώδικα

```
Point::Point(): x{0},y{0}
{
}

Point::Point(double a, double b) : x{ a },y{b}
{
}
```

Λειτουργία constructor

- Ο default constructor καλείται αυτόματα όταν δηλώνουμε ένα αντικείμενο αυτής της κλάσης
  - ΔΕΝ ΚΑΛΕΙΤΑΙ ΜΕΣΩ ΤΟΥ ΑΝΤΙΚΕΙΜΕΝΟΥ!!!
- Αν θέλουμε μπορούμε να καλέσουμε και όποιον άλλο θέλουμε
- Αν δημιουργήσουμε μη default constructor **ΠΡΕΠΕΙ απαραίτητα να δημιουργήσουμε και default**

```
int main()
{
    Point a, b{ 3,4 };
    Point *p;

    p = new Point{ 5,5 };

    a.print(); // (0,0)
    b.print();  // (3,4)
    p->print(); // (5,5)

}
```

Δημιουργία αντικειμένου

- Υπάρχουν διάφοροι τρόποι για να δημιουργήσουμε ένα αντικείμενο
  - Αν κάνουμε μόνο δήλωση καλείται ο default
  - Με την χρήση παρενθέσεων
  - Με το **list initialization** (αγκύλες)

```
Point a;
Point b(3, 4);
Point c{ 3,4 };

Point *p;

p = new Point;
p = new Point(5, 5);
p = new Point{ 5,5 };
```

Destructor

- Αντίστοιχα με τον constructor μπορεί να δημιουργηθεί και ο destructor. Ο destructor καλείται
  - Όταν τελειώνει η function / program
  - Το block στο οποίο δηλώθηκε το αντικείμενο
  - Με τον τελεστή delete
- Έχει το ίδιο όνομα με την κλάση με το ~ μπροστά
- Υπάρχει μόνο default

```
public:
    Point();
    Point(double a, double b);
    ~Point();
```

```
Point::~Point()
{
    cout << "Point destroyed\n";
}
```

Member functions : this & const

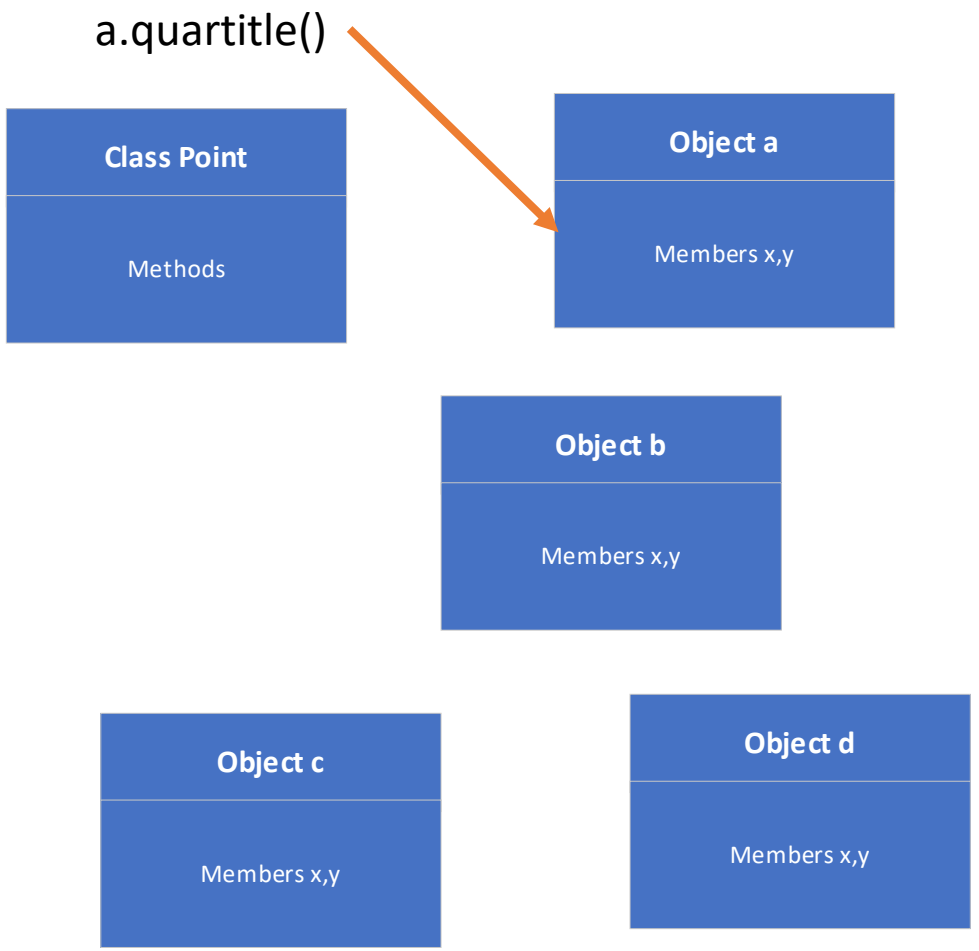
Const member functions

- Υπάρχουν μερικές member functions που δεν μεταβάλουν τα data members
- Αυτές εδώ μπορούμε να τις δηλώσουμε στον compiler γράφοντας την λέξη const στο τέλος της δήλωσης τους (πριν τον χαρακτήρα ;)

```
class Point
{
private:
    double x;
    double y;
public:
    int quartile() const;
    double get_x() const;
    double get_y() const;
    void set_x(double ax);
    void set_y(double ay);
    Point() = default;
    Point(double a, double b);
};
```

Όρισμα this (πρόβλημα)

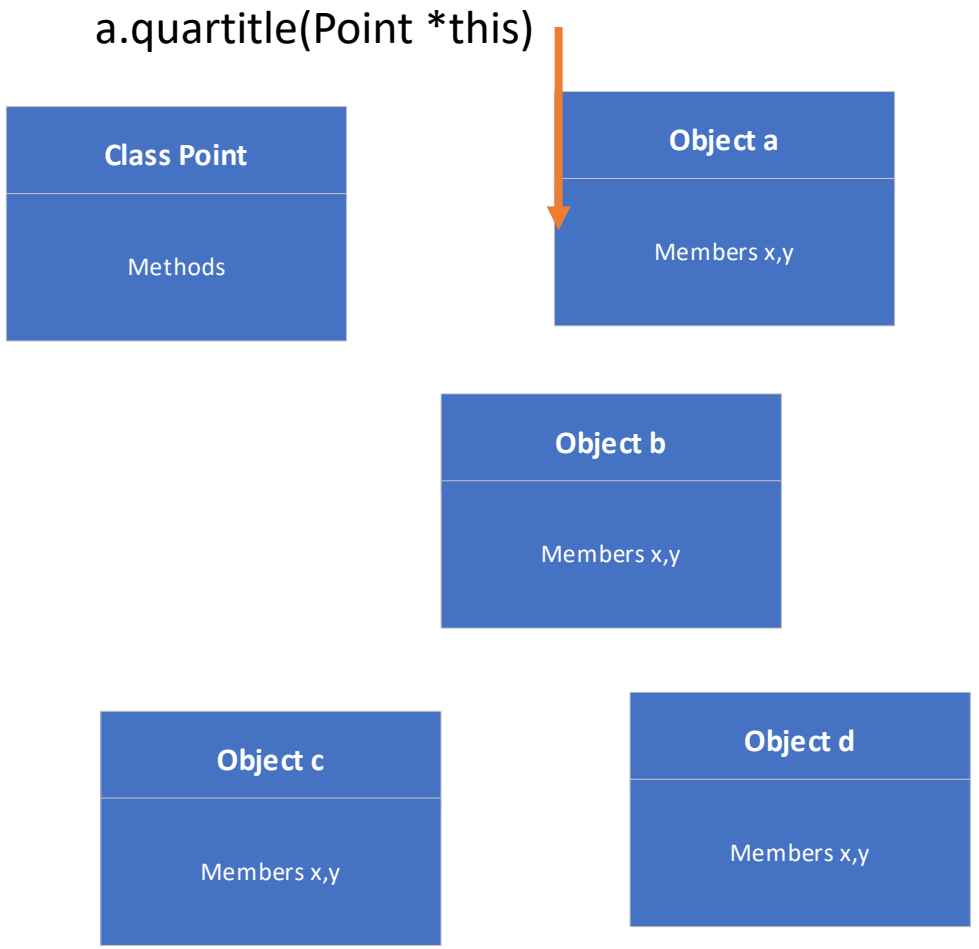
- Το κάθε αντικείμενο καταλαμβάνει χώρο στην μνήμη ΜΟΝΟ για τις μεταβλητές μέλη
- Οι συναρτήσεις μέλη αποθηκεύονται ΜΙΑ φορά για όλα τα αντικείμενα της κλάσης
- Πως γίνεται όμως ο κώδικας τους να έχει πρόσβαση στις μεταβλητές μέλη του συγκεκριμένου αντικειμένου που την κάλεσε;



Όρισμα this (λύση)

- Η κάθε συνάρτηση μέλος έχει και μία κρυφή παράμετρο (την this)
- Η this είναι ένας pointer στο αντικείμενο που κάλεσε την συγκεκριμένη μέθοδο

Μέσα στην συνάρτηση μέλος
x
Είναι ίδιο με
this->x



Call by reference

- Επειδή μπορεί ένα object να είναι μεγάλο σε μέγεθος καλύτερα είναι να γίνεται πέρασμα με reference
- Αν γνωρίζουμε ότι η συνάρτηση ΔΕΝ θα μεταβάλει το object τότε σαν const reference
- ΠΡΟΣΟΧΗ σε αυτή την περίπτωση αν χρησιμοποιεί μέσα μια μέθοδο του object τότε αυτή θα πρέπει απαραίτητα να είναι δηλωμένη const

```
void myfunc(const Point& a)
{
    cout << a.quartile() << endl;
}
```

Διάφορα

Κλάσεις που περιέχουν Pointer

- Μπορούν οι μεταβλητές μέλη μιας κλάσης να είναι pointers σε άλλα αντικείμενα
- Αυτού του είδους οι κλάσεις δημιουργούν αρκετά προβλήματα στην διαχείριση τους

```
class Triangle
{
private:
    Point* a, *b, *c;
public:
    Triangle() { a = b = c = nullptr; }
    Triangle(Point* x, Point* y, Point* z);

};
```

```
Point a{ 1,2 }, b{ 3,8 }, c{ -1,3 };
Triangle x{ &a,&b,&c };
Triangle y;

y = x;

x.a->get_x(); // what happes with y->x?
```

Container που περιέχει κλάσεις

- Μπορούμε να δημιουργήσουμε ένα container που να περιέχει μέσα κλάσεις που δημιουργήσαμε (map key ιδιαιτερότητα με τελεστή <)
- Μάλιστα τα στοιχεία που εισάγουμε μπορεί να είναι «ανώνυμα» αντικείμενα της κλάσης

```
vector<Point> sxima;
sxima.push_back(Point());
sxima.push_back(Point(3.5, 4.2));

for (auto z : sxima)
    cout << z.quartile() << endl;
```

Static members

Static members

- Όταν ένα member variable έχει δηλωθεί σαν static τότε αυτό είναι κοινό για όλη την κλάση
- Πρόσβαση σε αυτό μπορούμε να έχουμε είτε μέσα από αντικείμενο είτε από την ίδια την κλάση

```
private:
    double x;
    double y;
    static int count;
```

```
ka.count++;
Point::count++;
```