Περιεχόμενα
Βασικές έννοιες αντικειμενο-στρέφειας
Κλάσεις & Αντικείμενα
Δήλωση κλάσης
constructors
destructor
Υλοποίηση μεθόδων
Παράδειγμα προγράμματος
Δείκτες προς αντικείμενα
Δείκτης this
Εργαλεία
make
doxygen

Αντικειμενοστραφείς Γλώσσες Προγραμματισμού C++ / ROOT

Ιωάννης Παπαδόπουλος

Τμήμα Φυσικής, Πανεπιστήμιο Ιωαννίνων

Νοέμβριος 2018

Περιεχόμενα

Βασικές έννοιες αντικειμενο-στρέφειας

Κλάσεις & Αντικείμενα

Δήλωση κλάσης

constructors

destructor

Υλοποίηση μεθόδων

Παράδειγμα προγράμματος

Δείκτες προς αντικείμενα

Δείκτης this

Εργαλεία

make

doxygen

Περιεχόμενα

Περιεχόμενα

Περιεχόμενα
Βασικές έννοιες αντικειμενο-στρέφειας
Κλάσεις & Αντικείμενα
Δήλωση κλάσης
constructors
destructor
Υλοποίηση μεθόδων
Παράδειγμα προγράμματος
Δείκτες προς αντικείμενα
Δείκτης this
Εργαλεία
make
doxygen

1. Βασικές έννοιες αντικειμενοστρέφειας
2. Κλάσεις & Αντικείμενα
   - Δήλωση κλάσης
   - Μέθοδοι κατασκευής αντικειμένων (constructors)
   - Μέθοδος καταστροφής αντικειμένων (destructor)
   - Υλοποίηση μεθόδων κλάσης
   - Πρόγραμμα που χρησιμοποιεί την κλάση Ratio (testRatio.cpp)
   - Δείκτες προς αντικείμενα (testRatio2.cpp)
   - Δείκτης this (αυτο-αναφορά αντικειμένου)
3. Εργαλεία αυτοματοποίησης
   - Αυτοματοποίηση μεταγλώττισης με το πρόγραμμα make
   - Αυτόματη τεκμηρίωση πηγαίου κώδικα (πρόγραμμα doxygen)

Βασικές έννοιες αντικειμενοστρέφειας

Βασικές έννοιες αντικειμενοστρέφειας

Περιεχόμενα
Βασικές έννοιες αντικειμενο-στρέφειας
Κλάσεις & Αντικείμενα
Δήλωση κλάσης
constructors
destructor
Υλοποίηση μεθόδων
Παράδειγμα προγράμματος
Δείκτες προς αντικείμενα
Δείκτης this
Εργαλεία
make
doxygen

Αφαίρεση (abstraction)

- Αναγνώριση και εστίαση στις κοινές ιδιότητες των αντικειμένων και των διαδικασιών χειρισμού τους.
- Απλοποιημένη περιγραφή ενός περίπλοκου συστήματος.

Περιεχόμενα
Βασικές έννοιες αντικειμενο-στρέφειας
Κλάσεις & Αντικείμενα
Δήλωση κλάσης
constructors
destructor
Υλοποίηση μεθόδων
Παράδειγμα προγράμματος
Δείκτες προς αντικείμενα
Δείκτης this
Εργαλεία
make
doxygen

Βασικές έννοιες αντικειμενοστρέφειας

Αφαίρεση (abstraction)

- Αναγνώριση και εστίαση στις κοινές ιδιότητες των αντικειμένων και των διαδικασιών χειρισμού τους.
- Απλοποιημένη περιγραφή ενός περίπλοκου συστήματος.

Κελυφοποίηση (encapsulation)

- Απόκρυψη των λεπτομερειών υλοποίησης εντός αντικειμένου από το υπόλοιπο σύστημα.
- Κανένα σύστημα δεν πρέπει να εξαρτάται από τις λεπτομέρειες ενός άλλου συστήματος.

Βασικές έννοιες αντικειμενοστρέφειας

Αφαίρεση (abstraction)

- Αναγνώριση και εστίαση στις κοινές ιδιότητες των αντικειμένων και των διαδικασιών χειρισμού τους.
- Απλοποιημένη περιγραφή ενός περίπλοκου συστήματος.

Κελυφοποίηση (encapsulation)

- Απόκρυψη των λεπτομερειών υλοποίησης εντός αντικειμένου από το υπόλοιπο σύστημα.
- Κανένα σύστημα δεν πρέπει να εξαρτάται από τις λεπτομέρειες ενός άλλου συστήματος.

Δομικότητα (modularity)

- Κατάτμηση ενός σύνθετου συστήματος σε επιμέρους απλούστερα, για ευκολότερο χειρισμό της πολυπλοκότητας.
- Ανεξάρτητη ανάπτυξη και συντήρηση των επιμέρους συστημάτων.

Βασικές έννοιες αντικειμενοστρέφειας

Αφαίρεση (abstraction)

- Αναγνώριση και εστίαση στις κοινές ιδιότητες των αντικειμένων και των διαδικασιών χειρισμού τους.
- Απλοποιημένη περιγραφή ενός περίπλοκου συστήματος.

Κελυφοποίηση (encapsulation)

- Απόκρυψη των λεπτομερειών υλοποίησης εντός αντικειμένου από το υπόλοιπο σύστημα.
- Κανένα σύστημα δεν πρέπει να εξαρτάται από τις λεπτομέρειες ενός άλλου συστήματος.

Δομικότητα (modularity)

- Κατάτμηση ενός σύνθετου συστήματος σε επιμέρους απλούστερα, για ευκολότερο χειρισμό της πολυπλοκότητας.
- Ανεξάρτητη ανάπτυξη και συντήρηση των επιμέρους συστημάτων.

Ιεραρχία (hierarchy)

- Ταξινόμηση κλάσεων βάσει σχέσεων κληρονομικότητας, επέκτασης, αφαίρεσης, κ.ο.κ.
- Όσο πιο κάτω στην ιεραρχία βρίσκεται μία κλάση, τόσο πιο εξειδικευμένη είναι. Π.χ. Έμβιο ← Ζώο ← Σπονδυλωτό ← Θηλαστικό ← Σαρκοφάγο ← Αιλουροειδές ← Γάτα

Περιεχόμενα

Βασικές έννοιες αντικειμενο-στρέφειας

Κλάσεις & Αντικείμενα

Δήλωση κλάσης

constructors

destructor

Υλοποίηση μεθόδων

Παράδειγμα προγράμματος

Δείκτες προς αντικείμενα

Δείκτης this

Εργαλεία

make

doxygen

Κλάσεις & Αντικείμενα

Τι είναι αντικείμενο και τι κλάση;

Περιεχόμενα
Βασικές έννοιες αντικειμενο-στρέφειας
Κλάσεις & Αντικείμενα
Δήλωση κλάσης
constructors
destructor
Υλοποίηση μεθόδων
Παράδειγμα προγράμματος
Δείκτες προς αντικείμενα
Δείκτης this
Εργαλεία
make
doxygen

Αντικείμενο

Οποιαδήποτε οντότητα η οποία μπορεί να **αποθηκεύει** μέσα της **δεδομένα**, καθώς **και μεθόδους** για τη διαχείριση των δεδομένων αυτών.

Περιεχόμενα
Βασικές έννοιες αντικειμενο-στρέφειας
Κλάσεις & Αντικείμενα
Δήλωση κλάσης
constructors
destructor
Υλοποίηση μεθόδων
Παράδειγμα προγράμματος
Δείκτες προς αντικείμενα
Δείκτης this
Εργαλεία
make
doxygen

Τι είναι αντικείμενο και τι κλάση;

Αντικείμενο

Οποιαδήποτε οντότητα η οποία μπορεί να **αποθηκεύει** μέσα της **δεδομένα**, καθώς **και μεθόδους** για τη διαχείριση των δεδομένων αυτών.

Κλάση

- Είναι ένας **σύνθετος τύπος δεδομένων**, που ορίζεται από τον χρήστη και μπορεί να περιλαμβάνει στοιχεία όπως μεταβλητές βασικών τύπων, αντικείμενα άλλων κλάσεων, συναρτήσεις, ακόμη και τελεστές που (επαν-)υλοποιεί.
- **Περιγράφει μία κατηγορία αντικειμένων**, τα οποία έχουν ίδια δομή και συμπεριφορά
- Λειτουργεί ως **πρότυπο / μήτρα** για να κατασκευάζονται νέα αντικείμενα, δηλαδή **νέα στιγμιότυπα (instances)** της κλάσης.
- Μία κλάση μπορεί να χρησιμοποιηθεί μέσω της **διεπαφής** της (**interface**), που προκύπτει από τη δήλωσή της (class declaration), και πιο συγκεκριμένα από το κοινόχρηστο (public) μέρος αυτής.
- Η **υλοποίηση** της κλάσης, είναι εσωτερική της υπόθεση και αποκρύπτεται από όσους τη χρησιμοποιούν (κελυφοποίηση, **encapsulation**).

Δήλωση κλάσης (παράδειγμα: Ratio.h) (1/2)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();                // κατασκευή
7      Ratio(int);             // κατασκευή
8      Ratio(int,int);         // κατασκευή
9      ~Ratio();               // καταστροφή
10     int getNumerator();       // πρόσβαση
11     int getDenominator();     // πρόσβαση
12     void setNumerator(int);   // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();      // πραγματική τιμή
15     void invert();          // αντιστροφή
16     void print();           // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

- Μία κλάση δηλώνεται στη C++ με τη λέξη `class` και το όνομά της (π.χ. στη γραμμή 4: **Ratio**). Συνηθίζεται το όνομα μίας κλάσης να ξεκινά με κεφαλαίο γράμμα.

Δήλωση κλάσης (παράδειγμα: Ratio.h) (1/2)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();                // κατασκευή
7      Ratio(int);             // κατασκευή
8      Ratio(int,int);         // κατασκευή
9      ~Ratio();               // καταστροφή
10     int getNumerator();       // πρόσβαση
11     int getDenominator();     // πρόσβαση
12     void setNumerator(int);   // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();      // πραγματική τιμή
15     void invert();          // αντιστροφή
16     void print();           // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

- Μία κλάση δηλώνεται στη C++ με τη λέξη `class` και το όνομά της (π.χ. στη γραμμή 4: **Ratio**). Συνηθίζεται το όνομα μίας κλάσης να ξεκινά με κεφαλαίο γράμμα.
- Ακολουθούν οι **δηλώσεις των πεδίων** που αφορούν τη δομή της κλάσης (γραμμές 5-19):
  Περιλαμβάνονται μέθοδοι (συναρτήσεις) κατασκευής και καταστροφής, μέθοδοι πρόσβασης, άλλες μέθοδοι και δεδομένα (απλών τύπων ή και άλλων αντικειμένων).

Δήλωση κλάσης (παράδειγμα: Ratio.h) (1/2)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();                // κατασκευή
7      Ratio(int);             // κατασκευή
8      Ratio(int,int);         // κατασκευή
9      ~Ratio();               // καταστροφή
10     int getNumerator();       // πρόσβαση
11     int getDenominator();     // πρόσβαση
12     void setNumerator(int);   // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();      // πραγματική τιμή
15     void invert();          // αντιστροφή
16     void print();           // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

- Μία κλάση δηλώνεται στη C++ με τη λέξη `class` και το όνομά της (π.χ. στη γραμμή 4: **Ratio**). Συνηθίζεται το όνομα μίας κλάσης να ξεκινά με κεφαλαίο γράμμα.
- Ακολουθούν οι **δηλώσεις των πεδίων** που αφορούν τη δομή της κλάσης (γραμμές 5-19):
  Περιλαμβάνονται μέθοδοι (συναρτήσεις) κατασκευής και καταστροφής, μέθοδοι πρόσβασης, άλλες μέθοδοι και δεδομένα (απλών τύπων ή και άλλων αντικειμένων).
- Τα πεδία της κλάσης δηλώνονται εντός `{ };`
  Προσοχή στο τελικό ερωτηματικό...

Δήλωση κλάσης (παράδειγμα: Ratio.h) (1/2)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();                // κατασκευή
7      Ratio(int);             // κατασκευή
8      Ratio(int,int);         // κατασκευή
9      ~Ratio();               // καταστροφή
10     int getNumerator();       // πρόσβαση
11     int getDenominator();     // πρόσβαση
12     void setNumerator(int);   // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();      // πραγματική τιμή
15     void invert();          // αντιστροφή
16     void print();           // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

- Μία κλάση δηλώνεται στη C++ με τη λέξη `class` και το όνομά της (π.χ. στη γραμμή 4: **Ratio**). Συνηθίζεται το όνομα μίας κλάσης να ξεκινά με κεφαλαίο γράμμα.
- Ακολουθούν οι **δηλώσεις των πεδίων** που αφορούν τη δομή της κλάσης (γραμμές 5-19):
  Περιλαμβάνονται μέθοδοι (συναρτήσεις) κατασκευής και καταστροφής, μέθοδοι πρόσβασης, άλλες μέθοδοι και δεδομένα (απλών τύπων ή και άλλων αντικειμένων).
- Τα πεδία της κλάσης δηλώνονται εντός `{ };`
  Προσοχή στο τελικό ερωτηματικό...
- Η οδηγία `#pragma once` (γραμμή 2) προς τον προ-μεταγλωττιστή, εξασφαλίζει ότι το περιεχόμενο του αρχείου θα διαβαστεί μόνο μία φορά κατά τη μεταγλώττιση, ακόμη κι αν αυτό κληθεί περισσότερες φορές με εντολές `#include "Ratio.h"`.

Δήλωση κλάσης (παράδειγμα: Ratio.h) (2/2)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();                // κατασκευή
7      Ratio(int);             // κατασκευή
8      Ratio(int,int);         // κατασκευή
9      ~Ratio();               // καταστροφή
10     int getNumerator();       // πρόσβαση
11     int getDenominator();     // πρόσβαση
12     void setNumerator(int);   // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();      // πραγματική τιμή
15     void invert();          // αντιστροφή
16     void print();           // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

- Οι δεσμευμένες λέξεις/ετικέτες *public* και *private* χρησιμοποιούνται για τον **καθορισμό της κληρονομικότητας** των διαφόρων πεδίων, από τους χρήστες της κλάσης.

Δήλωση κλάσης (παράδειγμα: Ratio.h) (2/2)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();                // κατασκευή
7      Ratio(int);             // κατασκευή
8      Ratio(int,int);         // κατασκευή
9      ~Ratio();               // καταστροφή
10     int getNumerator();       // πρόσβαση
11     int getDenominator();     // πρόσβαση
12     void setNumerator(int);   // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();      // πραγματική τιμή
15     void invert();          // αντιστροφή
16     void print();           // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

- Οι δεσμευμένες λέξεις/ετικέτες *public* και *private* χρησιμοποιούνται για τον **καθορισμό της κληρονομικότητας** των διαφόρων πεδίων, από τους χρήστες της κλάσης.
- **public**:
  Ό,τι δηλώνεται ως κοινόχρηστο (public), είναι ορατό/προσβάσιμο από τον χρήστη της κλάσης.
  Το τμήμα αυτό αποτελεί τη **διεπαφή (interface)** της κλάσης.

Δήλωση κλάσης (παράδειγμα: Ratio.h) (2/2)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();                // κατασκευή
7      Ratio(int);             // κατασκευή
8      Ratio(int,int);         // κατασκευή
9      ~Ratio();               // καταστροφή
10     int getNumerator();       // πρόσβαση
11     int getDenominator();     // πρόσβαση
12     void setNumerator(int);   // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();      // πραγματική τιμή
15     void invert();          // αντιστροφή
16     void print();           // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

- Οι δεσμευμένες λέξεις/ετικέτες *public* και *private* χρησιμοποιούνται για τον **καθορισμό της κληρονομικότητας** των διαφόρων πεδίων, από τους χρήστες της κλάσης.
- **public**:
  Ό,τι δηλώνεται ως κοινόχρηστο (public), είναι ορατό/προσβάσιμο από τον χρήστη της κλάσης.
  Το τμήμα αυτό αποτελεί τη **διεπαφή (interface)** της κλάσης.
- **private**:
  Ό,τι δηλώνεται ως ιδιωτικό (private), δεν είναι προσβάσιμο από τον χρήστη της κλάσης.
  Στο διπλανό παράδειγμα, τα εσωτερικά δεδομένα της κλάσης μας (ο αριθμητής και ο παρανομαστής) είναι ιδιωτικά.

Δήλωση κλάσης (παράδειγμα: Ratio.h) (2/2)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();                // κατασκευή
7      Ratio(int);             // κατασκευή
8      Ratio(int,int);         // κατασκευή
9      ~Ratio();               // καταστροφή
10     int getNumerator();       // πρόσβαση
11     int getDenominator();     // πρόσβαση
12     void setNumerator(int);   // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();      // πραγματική τιμή
15     void invert();          // αντιστροφή
16     void print();           // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

Κανόνας:
Όλα τα δεδομένα μίας κλάσης πρέπει να είναι ιδιωτικά (private).

- Οι δεσμευμένες λέξεις/ετικέτες *public* και *private* χρησιμοποιούνται για τον **καθορισμό της κληρονομικότητας** των διαφόρων πεδίων, από τους χρήστες της κλάσης.
- **public**:
  Ό,τι δηλώνεται ως κοινόχρηστο (public), είναι ορατό/προσβάσιμο από τον χρήστη της κλάσης.
  Το τμήμα αυτό αποτελεί τη **διεπαφή (interface)** της κλάσης.
- **private**:
  Ό,τι δηλώνεται ως ιδιωτικό (private), δεν είναι προσβάσιμο από τον χρήστη της κλάσης.
  Στο διπλανό παράδειγμα, τα εσωτερικά δεδομένα της κλάσης μας (ο αριθμητής και ο παρανομαστής) είναι ιδιωτικά.
- Αν χρειάζεται, για την πρόσβαση (ανάγνωση/get, μεταβολή/set) στα ιδιωτικά δεδομένα, υλοποιούνται κατάλληλες μέθοδοι (π.χ. *getNumerator()* και *setNumerator()*).

Μέθοδοι κατασκευής αντικειμένων (constructors)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();              // κατασκευή
7      Ratio(int);           // κατασκευή
8      Ratio(int,int);       // κατασκευή
9      ~Ratio();             // καταστροφή
10     int getNumerator();       // πρόσβαση
11     int getDenominator();     // πρόσβαση
12     void setNumerator(int);   // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();    // πραγματική τιμή
15     void invert();        // αντιστροφή
16     void print();         // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

- **Κλάση**: πρότυπο/μήτρα για την κατασκευή αντικειμένων
- **Αντικείμενο**:
  Δημιουργείται καλώντας μία **μέθοδο κατασκευής (constructor) της κλάσης**.
  Καταστρέφεται όταν το αντικείμενο καλέσει τη **μέθοδο κατάστροφής** του (destructor).

- Η μέθοδος κατασκευής εκτελείται κατά τη δημιουργία αντικειμένων (στιγμιοτύπων) μίας κλάσης.

Μέθοδοι κατασκευής αντικειμένων (constructors)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();               // κατασκευή
7      Ratio(int);            // κατασκευή
8      Ratio(int,int);        // κατασκευή
9      ~Ratio();              // καταστροφή
10     int getNumerator();       // πρόσβαση
11     int getDenominator();     // πρόσβαση
12     void setNumerator(int);   // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();     // πραγματική τιμή
15     void invert();         // αντιστροφή
16     void print();          // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

- **Κλάση**: πρότυπο/μήτρα για την κατασκευή αντικειμένων
- **Αντικείμενο**:
  Δημιουργείται καλώντας μία **μέθοδο κατασκευής (constructor) της κλάσης**.
  Καταστρέφεται όταν το αντικείμενο καλέσει τη **μέθοδο κατάστροφής** του (destructor).

- Η μέθοδος κατασκευής εκτελείται κατά τη δημιουργία αντικειμένων (στιγμιοτύπων) μίας κλάσης.
- Η εξ ορισμού μέθοδος κατασκευής (default constructor) υπάρχει για κάθε κλάση, και δεν έχει κανένα όρισμα. Το όνομά της είναι ίδιο με το όνομα της κλάσης, π.χ. *Ratio()* .

Μέθοδοι κατασκευής αντικειμένων (constructors)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();                // κατασκευή
7      Ratio(int);             // κατασκευή
8      Ratio(int,int);         // κατασκευή
9      ~Ratio();               // καταστροφή
10     int getNumerator();        // πρόσβαση
11     int getDenominator();      // πρόσβαση
12     void setNumerator(int);    // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();      // πραγματική τιμή
15     void invert();          // αντιστροφή
16     void print();           // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

- **Κλάση**: πρότυπο/μήτρα για την κατασκευή αντικειμένων
- **Αντικείμενο**:
  Δημιουργείται καλώντας μία **μέθοδο κατασκευής (constructor) της κλάσης**.
  Καταστρέφεται όταν το αντικείμενο καλέσει τη **μέθοδο κατάστροφής** του (destructor).

- Η μέθοδος κατασκευής εκτελείται κατά τη δημιουργία αντικειμένων (στιγμιοτύπων) μίας κλάσης.
- Η εξ ορισμού μέθοδος κατασκευής (default constructor) υπάρχει για κάθε κλάση, και δεν έχει κανένα όρισμα. Το όνομά της είναι ίδιο με το όνομα της κλάσης, π.χ. *Ratio()* .
- Ανάλογα με τις ανάγκες που προκύπτουν από τον σχεδιασμό μίας κλάσης, μπορούν να υπάρχουν περισσότερες από μία μέθοδοι κατασκευής (constructors), με διαφορετική υπογραφή η κάθε μία (constructor overloading).

Μέθοδοι κατασκευής αντικειμένων (constructors)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();                // κατασκευή
7      Ratio(int);             // κατασκευή
8      Ratio(int,int);         // κατασκευή
9      ~Ratio();               // καταστροφή
10     int getNumerator();        // πρόσβαση
11     int getDenominator();      // πρόσβαση
12     void setNumerator(int);    // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();      // πραγματική τιμή
15     void invert();          // αντιστροφή
16     void print();           // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

- **Κλάση**: πρότυπο/μήτρα για την κατασκευή αντικειμένων
- **Αντικείμενο**:
  Δημιουργείται καλώντας μία **μέθοδο κατασκευής (constructor) της κλάσης**.
  Καταστρέφεται όταν το αντικείμενο καλέσει τη **μέθοδο κατάστροφής** του (destructor).

- Η μέθοδος κατασκευής εκτελείται κατά τη δημιουργία αντικειμένων (στιγμιοτύπων) μίας κλάσης.
- Η εξ ορισμού μέθοδος κατασκευής (default constructor) υπάρχει για κάθε κλάση, και δεν έχει κανένα όρισμα. Το όνομά της είναι ίδιο με το όνομα της κλάσης, π.χ. *Ratio()* .
- Ανάλογα με τις ανάγκες που προκύπτουν από τον σχεδιασμό μίας κλάσης, μπορούν να υπάρχουν περισσότερες από μία μέθοδοι κατασκευής (constructors), με διαφορετική υπογραφή η κάθε μία (constructor overloading).
- Οι μέθοδοι κατασκευής δεν έχουν τύπο και δεν επιστρέφουν τίποτε.

Μέθοδος καταστροφής αντικειμένων (destructor)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();                // κατασκευή
7      Ratio(int);             // κατασκευή
8      Ratio(int,int);         // κατασκευή
9      ~Ratio();               // καταστροφή
10     int getNumerator();        // πρόσβαση
11     int getDenominator();      // πρόσβαση
12     void setNumerator(int);    // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();      // πραγματική τιμή
15     void invert();          // αντιστροφή
16     void print();           // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

- **Κλάση**: πρότυπο/μήτρα για την κατασκευή αντικειμένων
- **Αντικείμενο**:
  Δημιουργείται καλώντας μία **μέθοδο κατασκευής (constructor) της κλάσης**.
  Καταστρέφεται όταν το αντικείμενο καλέσει τη **μέθοδο κατάστροφής** του (destructor).

- Η μέθοδος καταστροφής εκτελείται όταν ζητείται η διαγραφή ενός αντικειμένου (με την εντολή delete ή όταν το αντικείμενο βρεθεί εκτός εμβέλειας από την περιοχή ορισμού του). Συνήθως είναι άδεια. Έχει έννοια όταν πρέπει να ελευθερωθεί μνήμη που δεσμεύτηκε δυναμικά κατά τη διάρκεια ζωής του στιγμιοτύπου.

Μέθοδος καταστροφής αντικειμένων (destructor)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();                // κατασκευή
7      Ratio(int);             // κατασκευή
8      Ratio(int,int);         // κατασκευή
9      ~Ratio();               // καταστροφή
10     int getNumerator();        // πρόσβαση
11     int getDenominator();      // πρόσβαση
12     void setNumerator(int);    // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();      // πραγματική τιμή
15     void invert();          // αντιστροφή
16     void print();           // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

- **Κλάση**: πρότυπο/μήτρα για την κατασκευή αντικειμένων
- **Αντικείμενο**:
  Δημιουργείται καλώντας μία **μέθοδο κατασκευής (constructor) της κλάσης**.
  Καταστρέφεται όταν το αντικείμενο καλέσει τη **μέθοδο κατάστροφής** του (destructor).

- Η μέθοδος καταστροφής εκτελείται όταν ζητείται η διαγραφή ενός αντικειμένου (με την εντολή delete ή όταν το αντικείμενο βρεθεί εκτός εμβέλειας από την περιοχή ορισμού του). Συνήθως είναι άδεια. Έχει έννοια όταν πρέπει να ελευθερωθεί μνήμη που δεσμεύτηκε δυναμικά κατά τη διάρκεια ζωής του στιγμιοτύπου.
- Η μέθοδος καταστροφής, έχει το ίδιο όνομα με την κλάση με την πρόθεση μίας περισπωμένης, π.χ. *∼Ratio()*, και δεν έχει ορίσματα, οπότε υπάρχει μόνο μία (δεν μπορεί να υπερφορτωθεί).

Μέθοδος καταστροφής αντικειμένων (destructor)

```
1  // αρχείο Ratio.h
2  #pragma once
3
4  class Ratio {
5    public:
6      Ratio();                // κατασκευή
7      Ratio(int);             // κατασκευή
8      Ratio(int,int);         // κατασκευή
9      ~Ratio();               // καταστροφή
10     int getNumerator();        // πρόσβαση
11     int getDenominator();      // πρόσβαση
12     void setNumerator(int);    // πρόσβαση
13     void setDenominator(int); // πρόσβαση
14     double toDouble();      // πραγματική τιμή
15     void invert();          // αντιστροφή
16     void print();           // εκτύπωση
17   private:
18     int num;      // αριθμητής
19     int den;      // παρανομαστής
20 };
```

- **Κλάση**: πρότυπο/μήτρα για την κατασκευή αντικειμένων
- **Αντικείμενο**:
  Δημιουργείται καλώντας μία **μέθοδο κατασκευής (constructor) της κλάσης**.
  Καταστρέφεται όταν το αντικείμενο καλέσει τη **μέθοδο κατάστροφής** του (destructor).

- Η μέθοδος καταστροφής εκτελείται όταν ζητείται η διαγραφή ενός αντικειμένου (με την εντολή delete ή όταν το αντικείμενο βρεθεί εκτός εμβέλειας από την περιοχή ορισμού του). Συνήθως είναι άδεια. Έχει έννοια όταν πρέπει να ελευθερωθεί μνήμη που δεσμεύτηκε δυναμικά κατά τη διάρκεια ζωής του στιγμιοτύπου.
- Η μέθοδος καταστροφής, έχει το ίδιο όνομα με την κλάση με την πρόθεση μίας περισπωμένης, π.χ. *∼Ratio()*, και δεν έχει ορίσματα, οπότε υπάρχει μόνο μία (δεν μπορεί να υπερφορτωθεί).
- Η μέθοδος καταστροφής δεν έχει τύπο και δεν επιστρέφει τίποτε.

Υλοποίηση των μεθόδων της κλάσης (παράδειγμα: Ratio.cpp)

```cpp
// αρχείο Ratio.cpp
#include <iostream>
#include "Ratio.h"

Ratio::Ratio()             { num=0; den=1; } // κατασκευή
Ratio::Ratio(int i)        { num=i; den=1; } // κατασκευή
Ratio::Ratio(int i,int j) { num=i; den=j; } // κατασκευή
Ratio::~Ratio() { }    // άδεια μέθοδος καταστροφής

int Ratio::getNumerator() {          // πρόσβαση
  return num;
}
int Ratio::getDenominator() {        // πρόσβαση
  return den;
}
void Ratio::setNumerator(int i) {    // πρόσβαση
  num=i;
}
void Ratio::setDenominator(int i) { // πρόσβαση
  den=i;
}
double Ratio::toDouble() {     // μετατροπή
  return (double) num / (double) den;
}
void Ratio::invert() {         // αντιστροφή
   int tmp=num;
   num=den;
   den=tmp;
}
void Ratio::print() {          // εκτύπωση
  std::cout << num << "/" << den;
}
```

- Όλες οι μέθοδοι που δηλώθηκαν στο αρχείο Ratio.h, υλοποιούνται εντός του αρχείου Ratio.cpp.

Υλοποίηση των μεθόδων της κλάσης (παράδειγμα: Ratio.cpp)

```cpp
// αρχείο Ratio.cpp
#include <iostream>
#include "Ratio.h"

Ratio::Ratio()             { num=0; den=1; } // κατασκευή
Ratio::Ratio(int i)        { num=i; den=1; } // κατασκευή
Ratio::Ratio(int i,int j) { num=i; den=j; } // κατασκευή
Ratio::~Ratio() { }    // άδεια μέθοδος καταστροφής

int Ratio::getNumerator() {          // πρόσβαση
  return num;
}
int Ratio::getDenominator() {        // πρόσβαση
  return den;
}
void Ratio::setNumerator(int i) {    // πρόσβαση
  num=i;
}
void Ratio::setDenominator(int i) { // πρόσβαση
  den=i;
}
double Ratio::toDouble() {     // μετατροπή
  return (double) num / (double) den;
}
void Ratio::invert() {         // αντιστροφή
   int tmp=num;
   num=den;
   den=tmp;
}
void Ratio::print() {          // εκτύπωση
  std::cout << num << "/" << den;
}
```

- Όλες οι μέθοδοι που δηλώθηκαν στο αρχείο Ratio.h, υλοποιούνται εντός του αρχείου Ratio.cpp.
- Στο παράδειγμά μας υλοποιείται η κλάση Ratio με την οποία θέλουμε να περιγράψουμε και να χειριστούμε ρητούς αριθμούς.

Υλοποίηση των μεθόδων της κλάσης (παράδειγμα: Ratio.cpp)

```
1  // αρχείο Ratio.cpp
2  #include <iostream>
3  #include "Ratio.h"
4
5  Ratio::Ratio()             { num=0; den=1; } // κατασκευή
6  Ratio::Ratio(int i)        { num=i; den=1; } // κατασκευή
7  Ratio::Ratio(int i,int j) { num=i; den=j; } // κατασκευή
8  Ratio::~Ratio() { }    // άδεια μέθοδος καταστροφής
9
10 int Ratio::getNumerator() {          // πρόσβαση
11   return num;
12 }
13 int Ratio::getDenominator() {        // πρόσβαση
14   return den;
15 }
16 void Ratio::setNumerator(int i) {    // πρόσβαση
17   num=i;
18 }
19 void Ratio::setDenominator(int i) { // πρόσβαση
20   den=i;
21 }
22 double Ratio::toDouble() {     // μετατροπή
23   return (double) num / (double) den;
24 }
25 void Ratio::invert() {         // αντιστροφή
26    int tmp=num;
27    num=den;
28    den=tmp;
29 }
30 void Ratio::print() {          // εκτύπωση
31   std::cout << num << "/" << den;
32 }
```

- Όλες οι μέθοδοι που δηλώθηκαν στο αρχείο Ratio.h, υλοποιούνται εντός του αρχείου Ratio.cpp.
- Στο παράδειγμά μας υλοποιείται η κλάση Ratio με την οποία θέλουμε να περιγράψουμε και να χειριστούμε ρητούς αριθμούς.
- Με την ολοκλήρωση της υλοποίησης (implementation) των μεθόδων της κλάσης, είναι έτοιμη να χρησιμοποιηθεί, αφού μεταγλωττιστεί:

```
g++ -std=c++11 -c Ratio.cpp
```

Υλοποίηση των μεθόδων της κλάσης (παράδειγμα: Ratio.cpp)

```
1  // αρχείο Ratio.cpp
2  #include <iostream>
3  #include "Ratio.h"
4
5  Ratio::Ratio()             { num=0; den=1; } // κατασκευή
6  Ratio::Ratio(int i)        { num=i; den=1; } // κατασκευή
7  Ratio::Ratio(int i,int j) { num=i; den=j; } // κατασκευή
8  Ratio::~Ratio() { }    // άδεια μέθοδος καταστροφής
9
10 int Ratio::getNumerator() {          // πρόσβαση
11   return num;
12 }
13 int Ratio::getDenominator() {        // πρόσβαση
14   return den;
15 }
16 void Ratio::setNumerator(int i) {    // πρόσβαση
17   num=i;
18 }
19 void Ratio::setDenominator(int i) { // πρόσβαση
20   den=i;
21 }
22 double Ratio::toDouble() {     // μετατροπή
23   return (double) num / (double) den;
24 }
25 void Ratio::invert() {         // αντιστροφή
26    int tmp=num;
27    num=den;
28    den=tmp;
29 }
30 void Ratio::print() {          // εκτύπωση
31   std::cout << num << "/" << den;
32 }
```

- Όλες οι μέθοδοι που δηλώθηκαν στο αρχείο Ratio.h, υλοποιούνται εντός του αρχείου Ratio.cpp.
- Στο παράδειγμά μας υλοποιείται η κλάση Ratio με την οποία θέλουμε να περιγράψουμε και να χειριστούμε ρητούς αριθμούς.
- Με την ολοκλήρωση της υλοποίησης (implementation) των μεθόδων της κλάσης, είναι έτοιμη να χρησιμοποιηθεί, αφού μεταγλωττιστεί:

```
g++ -std=c++11 -c Ratio.cpp
```

- Από τη μεταγλώττιση προκύπτει το αρχείο Ratio.o αντικειμενικού κώδικα (object file), το οποίο μπορεί να χρησιμοποιηθεί αργότερα για την παραγωγή ενός τελικού εκτελέσιμου αρχείου.

Πρόγραμμα που χρησιμοποιεί την κλάση Ratio (testRatio.cpp)

```cpp
// αρχείο testRatio.cpp
#include <iostream>
#include "Ratio.h"
using namespace std;

int main() {
  int i=37;
  int j=11;
  Ratio r1;       // εξ'ορισμού μέθοδος κατασκευής: 0/1
  Ratio r2(15);  // κατασκευή με ένα όρισμα: 15/1
  Ratio r3(3,4); // κατασκευή με δύο ορίσματα: 3/4

  cout << "r1=";   r1.print();   cout << endl;
  cout << "r2=";   r2.print();   cout << endl;
  cout << "r3=";   r3.print();   cout << endl;

  r1.setNumerator(i);
  cout << "r1=";   r1.print();   cout << endl;
  r1.setDenominator(j);
  cout << "r1=";   r1.print();   cout << endl;

  return 0;
}
```

- Τα r1, r2 και r3 δηλώνονται ως αντικείμενα της κλάσης Ratio. Ανάλογα με τα ορίσματα που δίνονται, ο μεταγλωτιστής αποφασίζει κάθε φορά ποια μέθοδος κατασκευής θα χρησιμοποιηθεί.

Πρόγραμμα που χρησιμοποιεί την κλάση Ratio (testRatio.cpp)

```cpp
// αρχείο testRatio.cpp
#include <iostream>
#include "Ratio.h"
using namespace std;

int main() {
  int i=37;
  int j=11;
  Ratio r1;       // εξ'ορισμού μέθοδος κατασκευής: 0/1
  Ratio r2(15);  // κατασκευή με ένα όρισμα: 15/1
  Ratio r3(3,4); // κατασκευή με δύο ορίσματα: 3/4

  cout << "r1=";   r1.print();   cout << endl;
  cout << "r2=";   r2.print();   cout << endl;
  cout << "r3=";   r3.print();   cout << endl;

  r1.setNumerator(i);
  cout << "r1=";   r1.print();   cout << endl;
  r1.setDenominator(j);
  cout << "r1=";   r1.print();   cout << endl;

  return 0;
}
```

- Τα r1, r2 και r3 δηλώνονται ως αντικείμενα της κλάσης Ratio. Ανάλογα με τα ορίσματα που δίνονται, ο μεταγλωτιστής αποφασίζει κάθε φορά ποια μέθοδος κατασκευής θα χρησιμοποιηθεί.
- Για την κλήση μεθόδων ενός αντικειμένου, χρησιμοποιείται ο τελεστής επιλογής `'.'` (τελεία).

Πρόγραμμα που χρησιμοποιεί την κλάση Ratio (testRatio.cpp)

```cpp
// αρχείο testRatio.cpp
#include <iostream>
#include "Ratio.h"
using namespace std;

int main() {
  int i=37;
  int j=11;
  Ratio r1;       // εξ'ορισμού μέθοδος κατασκευής: 0/1
  Ratio r2(15);  // κατασκευή με ένα όρισμα: 15/1
  Ratio r3(3,4); // κατασκευή με δύο ορίσματα: 3/4

  cout << "r1=";   r1.print();   cout << endl;
  cout << "r2=";   r2.print();   cout << endl;
  cout << "r3=";   r3.print();   cout << endl;

  r1.setNumerator(i);
  cout << "r1=";   r1.print();   cout << endl;
  r1.setDenominator(j);
  cout << "r1=";   r1.print();   cout << endl;

  return 0;
}
```

- Τα r1, r2 και r3 δηλώνονται ως αντικείμενα της κλάσης Ratio. Ανάλογα με τα ορίσματα που δίνονται, ο μεταγλωτιστής αποφασίζει κάθε φορά ποια μέθοδος κατασκευής θα χρησιμοποιηθεί.
- Για την κλήση μεθόδων ενός αντικειμένου, χρησιμοποιείται ο τελεστής επιλογής `'.'` (τελεία).
- Χρησιμοποιώντας τις μεθόδους πρόσβασης, μπορούμε να λάβουμε ή να αλλάξουμε τις ιδιωτικές μεταβλητές ενός αντικειμένου. Στο παράδειγμα αλλάζουμε τον αριθμητή και τον παρανομαστή του r1.

Πρόγραμμα που χρησιμοποιεί την κλάση Ratio (testRatio.cpp)

```cpp
// αρχείο testRatio.cpp
#include <iostream>
#include "Ratio.h"
using namespace std;

int main() {
  int i=37;
  int j=11;
  Ratio r1;       // εξ'ορισμού μέθοδος κατασκευής: 0/1
  Ratio r2(15);  // κατασκευή με ένα όρισμα: 15/1
  Ratio r3(3,4); // κατασκευή με δύο ορίσματα: 3/4

  cout << "r1=";   r1.print();   cout << endl;
  cout << "r2=";   r2.print();   cout << endl;
  cout << "r3=";   r3.print();   cout << endl;

  r1.setNumerator(i);
  cout << "r1=";   r1.print();   cout << endl;
  r1.setDenominator(j);
  cout << "r1=";   r1.print();   cout << endl;

  return 0;
}
```

- Τα r1, r2 και r3 δηλώνονται ως αντικείμενα της κλάσης Ratio. Ανάλογα με τα ορίσματα που δίνονται, ο μεταγλωτιστής αποφασίζει κάθε φορά ποια μέθοδος κατασκευής θα χρησιμοποιηθεί.
- Για την κλήση μεθόδων ενός αντικειμένου, χρησιμοποιείται ο τελεστής επιλογής `'.'` (τελεία).
- Χρησιμοποιώντας τις μεθόδους πρόσβασης, μπορούμε να λάβουμε ή να αλλάξουμε τις ιδιωτικές μεταβλητές ενός αντικειμένου. Στο παράδειγμα αλλάζουμε τον αριθμητή και τον παρανομαστή του r1.
- Με την ολοκλήρωση της ανάπτυξης του προγράμματος, μπορούμε να το μεταγλωττίσουμε:

```
g++ -std=c++11 -c testRatio.cpp
```

Από τη μεταγλώττιση προκύπτει το αρχείο testRatio.o αντικειμενικού κώδικα (object file).

Πρόγραμμα που χρησιμοποιεί την κλάση Ratio (testRatio.cpp)

```cpp
// αρχείο testRatio.cpp
#include <iostream>
#include "Ratio.h"
using namespace std;

int main() {
  int i=37;
  int j=11;
  Ratio r1;       // εξ'ορισμού μέθοδος κατασκευής: 0/1
  Ratio r2(15);  // κατασκευή με ένα όρισμα: 15/1
  Ratio r3(3,4); // κατασκευή με δύο ορίσματα: 3/4

  cout << "r1=";   r1.print();   cout << endl;
  cout << "r2=";   r2.print();   cout << endl;
  cout << "r3=";   r3.print();   cout << endl;

  r1.setNumerator(i);
  cout << "r1=";   r1.print();   cout << endl;
  r1.setDenominator(j);
  cout << "r1=";   r1.print();   cout << endl;

  return 0;
}
```

- Τα r1, r2 και r3 δηλώνονται ως αντικείμενα της κλάσης Ratio. Ανάλογα με τα ορίσματα που δίνονται, ο μεταγλωτιστής αποφασίζει κάθε φορά ποια μέθοδος κατασκευής θα χρησιμοποιηθεί.
- Για την κλήση μεθόδων ενός αντικειμένου, χρησιμοποιείται ο τελεστής επιλογής `'.'` (τελεία).
- Χρησιμοποιώντας τις μεθόδους πρόσβασης, μπορούμε να λάβουμε ή να αλλάξουμε τις ιδιωτικές μεταβλητές ενός αντικειμένου. Στο παράδειγμα αλλάζουμε τον αριθμητή και τον παρανομαστή του r1.
- Με την ολοκλήρωση της ανάπτυξης του προγράμματος, μπορούμε να το μεταγλωττίσουμε:

```
g++ -std=c++11 -c testRatio.cpp
```

Από τη μεταγλώττιση προκύπτει το αρχείο testRatio.o αντικειμενικού κώδικα (object file).

Για να δημιουργήσουμε το τελικό εκτελέσιμο αρχείο, συνδυάζουμε τα object files που πλέον έχουμε:

```
g++ -std=c++11 Ratio.o testRatio.o -o testRatio.exe
```

Δείκτες προς αντικείμενα (testRatio2.cpp)

```cpp
// αρχείο testRatio2.cpp
#include <iostream>
#include "Ratio.h"
using namespace std;

int main() {
  Ratio r1;
  r1.setNumerator(7);
  r1.setDenominator(9);
  cout << "r1=";   r1.print();   cout << endl;

  Ratio *p1 = new Ratio;
  Ratio *p2;
  p2=&r1;

  cout << "*p1=";   (*p1).print();   cout << endl;
  cout << "*p2=";   (*p2).print();   cout << endl;
  // ισοδύναμα, χρησιμοποιώντας τον τελεστή "->" :
  cout << "*p1=";     p1->print();   cout << endl;
  cout << "*p2=";     p2->print();   cout << endl;

  // Δεν χρειάζομαι πλέον το p1 => ελευθέρωση μνήμης
  delete p1;

  p2->setNumerator(3);
  p2->setDenominator(8);
  cout << "*p2=";   p2->print();   cout << endl;

  return 0;
}
```

- Το r1 δηλώνεται ως αντικείμενο της κλάσης Ratio. Χρησιμοποιώντας τον τελεστή επιλογής μέλους ' . ', θέτουμε τον αριθμητή και τον παρανομαστή του.

Δείκτες προς αντικείμενα (testRatio2.cpp)

```cpp
// αρχείο testRatio2.cpp
#include <iostream>
#include "Ratio.h"
using namespace std;

int main() {
  Ratio r1;
  r1.setNumerator(7);
  r1.setDenominator(9);
  cout << "r1=";   r1.print();   cout << endl;

  Ratio *p1 = new Ratio;
  Ratio *p2;
  p2=&r1;

  cout << "*p1=";   (*p1).print();   cout << endl;
  cout << "*p2=";   (*p2).print();   cout << endl;
  // ισοδύναμα, χρησιμοποιώντας τον τελεστή "->" :
  cout << "*p1=";     p1->print();   cout << endl;
  cout << "*p2=";     p2->print();   cout << endl;

  // Δεν χρειάζομαι πλέον το p1 => ελευθέρωση μνήμης
  delete p1;

  p2->setNumerator(3);
  p2->setDenominator(8);
  cout << "*p2=";   p2->print();   cout << endl;

  return 0;
}
```

- Το r1 δηλώνεται ως αντικείμενο της κλάσης Ratio. Χρησιμοποιώντας τον τελεστή επιλογής μέλους ' . ', θέτουμε τον αριθμητή και τον παρανομαστή του.
- Τα p1 και p2 δηλώνονται ως δείκτες σε αντικείμενα Ratio. Το p1 αρχικοποιείται (γραμμή 12) δεσμεύοντας μνήμη (τελεστής new), ενώ το p2 δείχνει στη διεύθυνση του r1 (γραμμή 14).

Δείκτες προς αντικείμενα (testRatio2.cpp)

```
1  // αρχείο testRatio2.cpp
2  #include <iostream>
3  #include "Ratio.h"
4  using namespace std;
5
6  int main() {
7    Ratio r1;
8    r1.setNumerator(7);
9    r1.setDenominator(9);
10   cout << "r1=";   r1.print();   cout << endl;
11
12   Ratio *p1 = new Ratio;
13   Ratio *p2;
14   p2=&r1;
15
16   cout << "*p1=";   (*p1).print();   cout << endl;
17   cout << "*p2=";   (*p2).print();   cout << endl;
18   // ισοδύναμα, χρησιμοποιώντας τον τελεστή "->" :
19   cout << "*p1=";     p1->print();   cout << endl;
20   cout << "*p2=";     p2->print();   cout << endl;
21
22   // Δεν χρειάζομαι πλέον το p1 => ελευθέρωση μνήμης
23   delete p1;
24
25   p2->setNumerator(3);
26   p2->setDenominator(8);
27   cout << "*p2=";   p2->print();   cout << endl;
28
29   return 0;
30 }
```

- Το r1 δηλώνεται ως αντικείμενο της κλάσης Ratio. Χρησιμοποιώντας τον τελεστή επιλογής μέλους ' . ', θέτουμε τον αριθμητή και τον παρανομαστή του.
- Τα p1 και p2 δηλώνονται ως δείκτες σε αντικείμενα Ratio. Το p1 αρχικοποιείται (γραμμή 12) δεσμεύοντας μνήμη (τελεστής new), ενώ το p2 δείχνει στη διεύθυνση του r1 (γραμμή 14).
- Επιλογή μέλους για αντικείμενα: τελεστής ' . ', π.χ. r1.getNumerator()

Δείκτες προς αντικείμενα (testRatio2.cpp)

```
1  // αρχείο testRatio2.cpp
2  #include <iostream>
3  #include "Ratio.h"
4  using namespace std;
5
6  int main() {
7    Ratio r1;
8    r1.setNumerator(7);
9    r1.setDenominator(9);
10   cout << "r1=";   r1.print();   cout << endl;
11
12   Ratio *p1 = new Ratio;
13   Ratio *p2;
14   p2=&r1;
15
16   cout << "*p1=";   (*p1).print();   cout << endl;
17   cout << "*p2=";   (*p2).print();   cout << endl;
18   // ισοδύναμα, χρησιμοποιώντας τον τελεστή "->" :
19   cout << "*p1=";     p1->print();   cout << endl;
20   cout << "*p2=";     p2->print();   cout << endl;
21
22   // Δεν χρειάζομαι πλέον το p1 => ελευθέρωση μνήμης
23   delete p1;
24
25   p2->setNumerator(3);
26   p2->setDenominator(8);
27   cout << "*p2=";   p2->print();   cout << endl;
28
29   return 0;
30 }
```

- Το r1 δηλώνεται ως αντικείμενο της κλάσης Ratio. Χρησιμοποιώντας τον τελεστή επιλογής μέλους `'.'`, θέτουμε τον αριθμητή και τον παρανομαστή του.
- Τα p1 και p2 δηλώνονται ως δείκτες σε αντικείμενα Ratio. Το p1 αρχικοποιείται (γραμμή 12) δεσμεύοντας μνήμη (τελεστής new), ενώ το p2 δείχνει στη διεύθυνση του r1 (γραμμή 14).
- Επιλογή μέλους για αντικείμενα: τελεστής `'.'`, π.χ. r1.getNumerator()
- Επιλογή μέλους για δείκτες προς αντικείμενα: τελεστής `'->'`, π.χ. p1->print() $\equiv$ (*p1).print() Προτιμούμε τον τελεστή `'->'`

Δείκτες προς αντικείμενα (testRatio2.cpp)

```
1  // αρχείο testRatio2.cpp
2  #include <iostream>
3  #include "Ratio.h"
4  using namespace std;
5
6  int main() {
7    Ratio r1;
8    r1.setNumerator(7);
9    r1.setDenominator(9);
10   cout << "r1=";   r1.print();   cout << endl;
11
12   Ratio *p1 = new Ratio;
13   Ratio *p2;
14   p2=&r1;
15
16   cout << "*p1=";   (*p1).print();   cout << endl;
17   cout << "*p2=";   (*p2).print();   cout << endl;
18   // ισοδύναμα, χρησιμοποιώντας τον τελεστή "->" :
19   cout << "*p1=";     p1->print();   cout << endl;
20   cout << "*p2=";     p2->print();   cout << endl;
21
22   // Δεν χρειάζομαι πλέον το p1 => ελευθέρωση μνήμης
23   delete p1;
24
25   p2->setNumerator(3);
26   p2->setDenominator(8);
27   cout << "*p2=";   p2->print();   cout << endl;
28
29   return 0;
30 }
```

- Το r1 δηλώνεται ως αντικείμενο της κλάσης Ratio. Χρησιμοποιώντας τον τελεστή επιλογής μέλους '.', θέτουμε τον αριθμητή και τον παρανομαστή του.
- Τα p1 και p2 δηλώνονται ως δείκτες σε αντικείμενα Ratio. Το p1 αρχικοποιείται (γραμμή 12) δεσμεύοντας μνήμη (τελεστής new), ενώ το p2 δείχνει στη διεύθυνση του r1 (γραμμή 14).
- Επιλογή μέλους για αντικείμενα: τελεστής '.', π.χ. r1.getNumerator()
- Επιλογή μέλους για δείκτες προς αντικείμενα: τελεστής '->', π.χ. p1->print() ≡ (*p1).print() Προτιμούμε τον τελεστή '->'
- Με την ολοκλήρωση της ανάπτυξης του προγράμματος, μπορούμε να το μεταγλωττίσουμε:

```
g++ -std=c++11 -c testRatio2.cpp
```

Από τη μεταγλώττιση προκύπτει το αρχείο testRatio2.o αντικειμενικού κώδικα (object file).

Δείκτες προς αντικείμενα (testRatio2.cpp)

```
1  // αρχείο testRatio2.cpp
2  #include <iostream>
3  #include "Ratio.h"
4  using namespace std;
5
6  int main() {
7    Ratio r1;
8    r1.setNumerator(7);
9    r1.setDenominator(9);
10   cout << "r1=";   r1.print();   cout << endl;
11
12   Ratio *p1 = new Ratio;
13   Ratio *p2;
14   p2=&r1;
15
16   cout << "*p1=";   (*p1).print();   cout << endl;
17   cout << "*p2=";   (*p2).print();   cout << endl;
18   // ισοδύναμα, χρησιμοποιώντας τον τελεστή "->" :
19   cout << "*p1=";     p1->print();   cout << endl;
20   cout << "*p2=";     p2->print();   cout << endl;
21
22   // Δεν χρειάζομαι πλέον το p1 => ελευθέρωση μνήμης
23   delete p1;
24
25   p2->setNumerator(3);
26   p2->setDenominator(8);
27   cout << "*p2=";   p2->print();   cout << endl;
28
29   return 0;
30 }
```

Για να δημιουργήσουμε το τελικό εκτελέσιμο αρχείο, συνδυάζουμε τα object files που πλέον έχουμε:

```
g++ -std=c++11 Ratio.o testRatio2.o -o testRatio2.exe
```

- Το r1 δηλώνεται ως αντικείμενο της κλάσης Ratio. Χρησιμοποιώντας τον τελεστή επιλογής μέλους `'.'`, θέτουμε τον αριθμητή και τον παρανομαστή του.
- Τα p1 και p2 δηλώνονται ως δείκτες σε αντικείμενα Ratio. Το p1 αρχικοποιείται (γραμμή 12) δεσμεύοντας μνήμη (τελεστής new), ενώ το p2 δείχνει στη διεύθυνση του r1 (γραμμή 14).
- Επιλογή μέλους για αντικείμενα: τελεστής `'.'`, π.χ. r1.getNumerator()
- Επιλογή μέλους για δείκτες προς αντικείμενα: τελεστής `'->'`, π.χ. p1->print() ≡ (*p1).print() Προτιμούμε τον τελεστή `'->'`
- Με την ολοκλήρωση της ανάπτυξης του προγράμματος, μπορούμε να το μεταγλωττίσουμε:

```
g++ -std=c++11 -c testRatio2.cpp
```

Από τη μεταγλώττιση προκύπτει το αρχείο testRatio2.o αντικειμενικού κώδικα (object file).

Δείκτης this (αυτο-αναφορά αντικειμένου)

```cpp
#include <iostream>      // πρόγραμμα test.cpp
using namespace std;

class TestClass {
   public:
     ~TestClass();
     TestClass();
     TestClass(double,double,double);
     void print();
   private:
     double x, y, z;
};

TestClass::~TestClass() {}
TestClass::TestClass() {x=y=z=0;}
TestClass::TestClass(double x=0,double y=0,double z=0) {
  this->x = x;  // this->x : ιδιωτικό μέλος x
                //       x : 1ο όρισμα
  this->y = y;  // this->y : ιδιωτικό μέλος y
                //       y : 2ο όρισμα
  this->z = z;  // this->z : ιδιωτικό μέλος z
                //       z : 3ο όρισμα
}
void TestClass::print() {
  cout << "("<<x<<","<<y<<","<<z<<")";
}

int main() {
  TestClass a(1,2,3);
  TestClass b(1);
  TestClass z(3,5);
  cout << "point1=";   a.print();   cout << endl;
  cout << "point2=";   b.print();   cout << endl;
  cout << "point7=";   z.print();   cout << endl;
}
```

- Όταν ένα αντικείμενο καλεί μία μέθοδό του (μη στατική), περνά σιωπηλά ως όρισμα ένας δείκτης (με όνομα **this**) προς το ίδιο το αντικείμενο.

Δείκτης this (αυτο-αναφορά αντικειμένου)

```cpp
#include <iostream>      // πρόγραμμα test.cpp
using namespace std;

class TestClass {
   public:
     ~TestClass();
     TestClass();
     TestClass(double,double,double);
     void print();
   private:
     double x, y, z;
};

TestClass::~TestClass() {}
TestClass::TestClass() {x=y=z=0;}
TestClass::TestClass(double x=0,double y=0,double z=0) {
  this->x = x;  // this->x : ιδιωτικό μέλος x
                //       x : 1ο όρισμα
  this->y = y;  // this->y : ιδιωτικό μέλος y
                //       y : 2ο όρισμα
  this->z = z;  // this->z : ιδιωτικό μέλος z
                //       z : 3ο όρισμα
}
void TestClass::print() {
  cout << "("<<x<<","<<y<<","<<z<<")";
}

int main() {
  TestClass a(1,2,3);
  TestClass b(1);
  TestClass z(3,5);
  cout << "point1=";   a.print();   cout << endl;
  cout << "point2=";   b.print();   cout << endl;
  cout << "point7=";   z.print();   cout << endl;
}
```

- Όταν ένα αντικείμενο καλεί μία μέθοδό του (μη στατική), περνά σιωπηλά ως όρισμα ένας δείκτης (με όνομα **this**) προς το ίδιο το αντικείμενο.
- Ο δείκτης **this** είναι διαθέσιμος εντός της καλούμενης μεθόδου, και μπορεί για παράδειγμα να χρησιμοποοιηθεί για την πρόσβαση στα ιδιωτικά (private) δεδομένα του εν λόγω αντικειμένου.

Δείκτης this (αυτο-αναφορά αντικειμένου)

```cpp
#include <iostream>      // πρόγραμμα test.cpp
using namespace std;

class TestClass {
   public:
     ~TestClass();
     TestClass();
     TestClass(double,double,double);
     void print();
   private:
     double x, y, z;
};

TestClass::~TestClass() {}
TestClass::TestClass() {x=y=z=0;}
TestClass::TestClass(double x=0,double y=0,double z=0) {
  this->x = x;  // this->x : ιδιωτικό μέλος x
                //         x : 1ο όρισμα
  this->y = y;  // this->y : ιδιωτικό μέλος y
                //         y : 2ο όρισμα
  this->z = z;  // this->z : ιδιωτικό μέλος z
                //         z : 3ο όρισμα
}
void TestClass::print() {
  cout << "("<<x<<","<<y<<","<<z<<")";
}

int main() {
  TestClass a(1,2,3);
  TestClass b(1);
  TestClass z(3,5);
  cout << "point1=";   a.print();   cout << endl;
  cout << "point2=";   b.print();   cout << endl;
  cout << "point7=";   z.print();   cout << endl;
}
```

- Όταν ένα αντικείμενο καλεί μία μέθοδό του (μη στατική), περνά σιωπηλά ως όρισμα ένας δείκτης (με όνομα **this**) προς το ίδιο το αντικείμενο.
- Ο δείκτης **this** είναι διαθέσιμος εντός της καλούμενης μεθόδου, και μπορεί για παράδειγμα να χρησιμοποοιηθεί για την πρόσβαση στα ιδιωτικά (private) δεδομένα του εν λόγω αντικειμένου.
- Στο διπλανό παράδειγμα, η μέθοδος κατασκευής που υλοποιείται στις γραμμές 16-23 δέχεται ως ορίσματα τα x, y και z. Όμως τα ονόματά τους είναι συνώνυμα των ιδιωτικών δεδομένων x, y και z του αντικειμένου.

Δείκτης this (αυτο-αναφορά αντικειμένου)

```cpp
#include <iostream>      // πρόγραμμα test.cpp
using namespace std;

class TestClass {
   public:
     ~TestClass();
     TestClass();
     TestClass(double,double,double);
     void print();
   private:
     double x, y, z;
};

TestClass::~TestClass() {}
TestClass::TestClass() {x=y=z=0;}
TestClass::TestClass(double x=0,double y=0,double z=0) {
  this->x = x;  // this->x : ιδιωτικό μέλος x
                //          x : 1ο όρισμα
  this->y = y;  // this->y : ιδιωτικό μέλος y
                //          y : 2ο όρισμα
  this->z = z;  // this->z : ιδιωτικό μέλος z
                //          z : 3ο όρισμα
}
void TestClass::print() {
  cout << "("<<x<<","<<y<<","<<z<<")";
}

int main() {
  TestClass a(1,2,3);
  TestClass b(1);
  TestClass z(3,5);
  cout << "point1=";   a.print();   cout << endl;
  cout << "point2=";   b.print();   cout << endl;
  cout << "point7=";   z.print();   cout << endl;
}
```

- Όταν ένα αντικείμενο καλεί μία μέθοδό του (μη στατική), περνά σιωπηλά ως όρισμα ένας δείκτης (με όνομα **this**) προς το ίδιο το αντικείμενο.
- Ο δείκτης **this** είναι διαθέσιμος εντός της καλούμενης μεθόδου, και μπορεί για παράδειγμα να χρησιμοποοιηθεί για την πρόσβαση στα ιδιωτικά (private) δεδομένα του εν λόγω αντικειμένου.
- Στο διπλανό παράδειγμα, η μέθοδος κατασκευής που υλοποιείται στις γραμμές 16-23 δέχεται ως ορίσματα τα x, y και z. Όμως τα ονόματά τους είναι συνώνυμα των ιδιωτικών δεδομένων x, y και z του αντικειμένου.
- Ο δείκτης this χρησιμοποιείται στην περίπτωση αυτή για να ξεχωρίσει τα ιδιωτικά δεδομένα από τα ονόματα των ορισμάτων.

Δείκτης this (αυτο-αναφορά αντικειμένου)

```cpp
#include <iostream>      // πρόγραμμα test.cpp
using namespace std;

class TestClass {
   public:
     ~TestClass();
     TestClass();
     TestClass(double,double,double);
     void print();
   private:
     double x, y, z;
};

TestClass::~TestClass() {}
TestClass::TestClass() {x=y=z=0;}
TestClass::TestClass(double x=0,double y=0,double z=0) {
  this->x = x;  // this->x : ιδιωτικό μέλος x
                //          x : 1ο όρισμα
  this->y = y;  // this->y : ιδιωτικό μέλος y
                //          y : 2ο όρισμα
  this->z = z;  // this->z : ιδιωτικό μέλος z
                //          z : 3ο όρισμα
}
void TestClass::print() {
  cout << "("<<x<<","<<y<<","<<z<<")";
}

int main() {
  TestClass a(1,2,3);
  TestClass b(1);
  TestClass z(3,5);
  cout << "point1=";   a.print();   cout << endl;
  cout << "point2=";   b.print();   cout << endl;
  cout << "point7=";   z.print();   cout << endl;
}
```

- Όταν ένα αντικείμενο καλεί μία μέθοδό του (μη στατική), περνά σιωπηλά ως όρισμα ένας δείκτης (με όνομα **this**) προς το ίδιο το αντικείμενο.
- Ο δείκτης **this** είναι διαθέσιμος εντός της καλούμενης μεθόδου, και μπορεί για παράδειγμα να χρησιμοποιηθεί για την πρόσβαση στα ιδιωτικά (private) δεδομένα του εν λόγω αντικειμένου.
- Στο διπλανό παράδειγμα, η μέθοδος κατασκευής που υλοποιείται στις γραμμές 16-23 δέχεται ως ορίσματα τα x, y και z. Όμως τα ονόματά τους είναι συνώνυμα των ιδιωτικών δεδομένων x, y και z του αντικειμένου.
- Ο δείκτης this χρησιμοποιείται στην περίπτωση αυτή για να ξεχωρίσει τα ιδιωτικά δεδομένα από τα ονόματα των ορισμάτων.
- Για τη μεταγλώττιση του διπλανού προγράμματος:

```
g++ -std=c++11 test.cpp -o test.exe
```

Εργαλεία αυτοματοποίησης

Αυτοματοποίηση μεταγλώττισης με το πρόγραμμα make

- Όταν ένα project περιλαμβάνει πολλά αρχεία πηγαίου κώδικα, αν γίνει κάποια αλλαγή σε ένα μόνο από αυτά, δεν είναι ανάγκη να μεταγλωτιστούν όλα τα αρχεία πηγαίου κώδικα, αλλά μόνον όσα επηρεάζονται από την αλλαγή που έγινε.

περιεχόμενα αρχείου makefile

```
# αρχείο makefile
testRatio.exe: Ratio.o testRatio.o
    g++ -o testRatio.exe Ratio.o testRatio.o
Ratio.o: Ratio.cpp Ratio.h
    g++ -g -c Ratio.cpp
testRatio.o: testRatio.cpp Ratio.h
    g++ -g -c testRatio.cpp
clean:
    rm -f testRatio.exe *~ Ratio.o testRatio.o
```

Περιεχόμενα
Βασικές έννοιες αντικειμενο-στρέφειας
Κλάσεις & Αντικείμενα
Δήλωση κλάσης
constructors
destructor
Υλοποίηση μεθόδων
Παράδειγμα προγράμματος
Δείκτες προς αντικείμενα
Δείκτης this
Εργαλεία
make
doxygen

Αυτοματοποίηση μεταγλώττισης με το πρόγραμμα make

περιεχόμενα αρχείου makefile

```
# αρχείο makefile
testRatio.exe: Ratio.o testRatio.o
    g++ -o testRatio.exe Ratio.o testRatio.o
Ratio.o: Ratio.cpp Ratio.h
    g++ -g -c Ratio.cpp
testRatio.o: testRatio.cpp Ratio.h
    g++ -g -c testRatio.cpp
clean:
    rm -f testRatio.exe *~ Ratio.o testRatio.o
```

- Όταν ένα project περιλαμβάνει πολλά αρχεία πηγαίου κώδικα, αν γίνει κάποια αλλαγή σε ένα μόνο από αυτά, δεν είναι ανάγκη να μεταγλωτιστούν όλα τα αρχεία πηγαίου κώδικα, αλλά μόνον όσα επηρεάζονται από την αλλαγή που έγινε.
- Τα περισσότερα αντικειμενικά αρχεία μπορούν να μείνουν ως είχαν, και να συνδυαστούν εκ νέου με όσα άλλαξαν για την παραγωγή του νέου εκτελέσιμου αρχείου.

Περιεχόμενα
Βασικές έννοιες αντικειμενο-στρέφειας
Κλάσεις & Αντικείμενα
Δήλωση κλάσης
constructors
destructor
Υλοποίηση μεθόδων
Παράδειγμα προγράμματος
Δείκτες προς αντικείμενα
Δείκτης this
Εργαλεία
make
doxygen

Αυτοματοποίηση μεταγλώττισης με το πρόγραμμα make

περιεχόμενα αρχείου makefile

```
# αρχείο makefile
testRatio.exe: Ratio.o testRatio.o
    g++ -o testRatio.exe Ratio.o testRatio.o
Ratio.o: Ratio.cpp Ratio.h
    g++ -g -c Ratio.cpp
testRatio.o: testRatio.cpp Ratio.h
    g++ -g -c testRatio.cpp
clean:
    rm -f testRatio.exe *~ Ratio.o testRatio.o
```

- Όταν ένα project περιλαμβάνει πολλά αρχεία πηγαίου κώδικα, αν γίνει κάποια αλλαγή σε ένα μόνο από αυτά, δεν είναι ανάγκη να μεταγλωτιστούν όλα τα αρχεία πηγαίου κώδικα, αλλά μόνον όσα επηρεάζονται από την αλλαγή που έγινε.
- Τα περισσότερα αντικειμενικά αρχεία μπορούν να μείνουν ως είχαν, και να συνδυαστούν εκ νέου με όσα άλλαξαν για την παραγωγή του νέου εκτελέσιμου αρχείου.
- Η διαδικασία αυτή μπορεί να καταστεί αρκετά επίπονη, και το πρόγραμμα **make** αναλαμβάνει να απλοποιήσει τη διαδικασία. Αρκεί να γράψουμε ένα **makefile** με τους κανόνες που διέπουν τη μεταγλώττιση του project.
  Το πρόγραμμα make αναλαμβάνει από κει και πέρα να μεταγλωττίσει ό,τι χρειάζεται και να φτιάξει το νέο εκτελέσιμο.

Αυτόματη τεκμηρίωση πηγαίου κώδικα (πρόγραμμα doxygen)

- Το λογισμικό doxygen σαρώνει όλα τα αρχεία πηγαίου κώδικα και δημιουργεί αυτομάτως τεκμηρίωση (documentation) σχετικά με τη δομή και την ιεραρχία ενός project.

Αυτόματη τεκμηρίωση πηγαίου κώδικα (πρόγραμμα doxygen)

- Το λογισμικό doxygen σαρώνει όλα τα αρχεία πηγαίου κώδικα και δημιουργεί αυτομάτως τεκμηρίωση (documentation) σχετικά με τη δομή και την ιεραρχία ενός project.
- Η τεκμηρίωση μπορεί να ληφθεί σε διάφορες μορφές, π.χ. στη μορφή ιστοσελίδας, σε μορφή LaTeX, PDF κ.ο.κ.

Αυτόματη τεκμηρίωση πηγαίου κώδικα (πρόγραμμα doxygen)

- Το λογισμικό doxygen σαρώνει όλα τα αρχεία πηγαίου κώδικα και δημιουργεί αυτομάτως τεκμηρίωση (documentation) σχετικά με τη δομή και την ιεραρχία ενός project.
- Η τεκμηρίωση μπορεί να ληφθεί σε διάφορες μορφές, π.χ. στη μορφή ιστοσελίδας, σε μορφή LaTeX, PDF κ.ο.κ.
- Υπάρχει δυνατότητα πληρέστερης και λεπτομερέστερης τεκμηρίωσης, μέσω της εισαγωγής ειδικών σχολίων εντός του πηγαίου κώδικα (χρησιμοποιώντας συγκεκριμένη σήμανση/markup)

Περιεχόμενα

Βασικές έννοιες αντικειμενο-στρέφειας

Κλάσεις & Αντικείμενα

Δήλωση κλάσης

constructors

destructor

Υλοποίηση μεθόδων

Παράδειγμα προγράμματος

Δείκτες προς αντικείμενα

Δείκτης this

Εργαλεία

make

doxygen

Αυτόματη τεκμηρίωση πηγαίου κώδικα (πρόγραμμα doxygen)

- Το λογισμικό doxygen σαρώνει όλα τα αρχεία πηγαίου κώδικα και δημιουργεί αυτομάτως τεκμηρίωση (documentation) σχετικά με τη δομή και την ιεραρχία ενός project.
- Η τεκμηρίωση μπορεί να ληφθεί σε διάφορες μορφές, π.χ. στη μορφή ιστοσελίδας, σε μορφή LaTeX, PDF κ.ο.κ.
- Υπάρχει δυνατότητα πληρέστερης και λεπτομερέστερης τεκμηρίωσης, μέσω της εισαγωγής ειδικών σχολίων εντός του πηγαίου κώδικα (χρησιμοποιώντας συγκεκριμένη σήμανση/markup)
- Παραδείγματα μεγάλων projects που χρησιμοποιούν το doxygen για την τεκμηρίωσή τους:
  - Κλάση TCanvas του λογισμικού ROOT
  - Κλάση TF1 του λογισμικού ROOT
  - Κλάση TH1 του λογισμικού ROOT
  - Κλάση G4ParticleGun του λογισμικού GEANT4
  - Κλάση G4PhotoElectricEffect του λογισμικού GEANT4