

Κακαρόντζας Γιώργος

ΚΛΆΣΕΙΣ, ΑΝΤΙΚΕΪΜΕΝΑ ΚΑΙ ΙΔΙΌΤΗΤΕΣ ΣΤΗ C#

Τι θα δούμε σε αυτό το μάθημα;

- ◉ Ορισμός μιας κλάσης ως συλλογής συσχετιζόμενων μεθόδων και πεδίων δεδομένων
- ◉ Έλεγχος πρόσβασης με την χρήση προσδιοριστών πρόσβασης
- ◉ Δημιουργία αντικειμένων με την χρήση κατασκευαστών και της λέξης-κλειδί new
- ◉ Κατασκευή των δικών σας κατασκευαστών
- ◉ Μερικές (partial) κλάσεις
- ◉ Δημιουργία στατικών μεθόδων και δεδομένων
- ◉ Ανώνυμες κλάσεις
- ◉ Δημιουργία και πρόσβαση ιδιοτήτων κλάσεων

Ορισμός μιας κλάσης

- Στη C# χρησιμοποιείτε την λέξη-κλειδί `class` για να ορίσετε μια κλάση
- Παράδειγμα η κλάση `Circle`

```
class Circle
{
    int radius;
    double Area()
    {
        return Math.PI * radius * radius;
    }
}
```

Τι περιέχει μία κλάση

- Μία κλάση περιέχει (ενθυλακώνει) κάποια δεδομένα και μεθόδους που επενεργούν στα δεδομένα αυτά.
- Η πρόθεσή μας όταν σχεδιάζουμε μία κλάση είναι να περιορίσουμε όσο τον δυνατόν περισσότερο την εξάρτησή της με άλλες κλάσεις. Επομένως μια καλή αρχή είναι πως τοποθετούμε τα δεδομένα μαζί με τις μεθόδους που τα επεξεργάζονται στην ίδια κλάση.
- Επίσης κάτι σημαντικό είναι πως οι κλάσεις αντιστοιχούν ευθέως σε έννοιες που έχουμε και στον πραγματικό κόσμο (π.χ. Πελάτης, Λογαριασμός, Συναλλαγή κλπ.)

Προσδιοριστές πρόσβασης

- Στην δήλωση της κλάσης Circle δεν δηλώσαμε προσδιοριστή πρόσβασης για το πεδίο radius ή για την μέθοδο Area.
- Εξ ορισμού όταν δεν δηλώνεται προσδιοριστής πρόσβασης υπονοείται ο private (ιδιωτική πρόσβαση) που σημαίνει πως το πεδίο ή η μέθοδος είναι προσπελάσιμη μόνο εσωτερικά από τις μεθόδους της ίδιας κλάσης και όχι από άλλες κλάσεις.
- Συνήθως τα πεδία δηλώνονται ως ιδιωτικά και οι μέθοδοι (όσες χρειάζεται να κληθούν από άλλες κλάσεις) ως δημόσιες (public). Για τα πεδία που χρειάζεται να έχουμε την δυνατότητα ανάκτησης ή μεταβολής της τιμής τους χρησιμοποιούμε ιδιότητες (properties) τις οποίες θα συζητήσουμε αργότερα.
- Υπάρχουν και άλλοι προσδιοριστές πρόσβασης που θα συζητήσουμε σε επόμενα μαθήματα.

Προσδιοριστές πρόσβασης

- Στο παράδειγμα το πεδίο `radius` δεν μπορεί να χρησιμοποιηθεί απευθείας από άλλες κλάσεις (γιατί είναι `private`) αλλά η μέθοδος `Area` μπορεί να κληθεί από άλλες κλάσεις (γιατί είναι `public`):

```
class Circle
{
    private int radius;
    public double Area()
    {
        return Math.PI * radius * radius;
    }
}
```

Αρχικοποίηση αντικειμένων

- Τα πεδία μιας κλάσης (όπως η radius) αρχικοποιούνται σε κάποιες εξ ορισμού τιμές ανάλογα με τον τύπο τους (π.χ. σε 0, false ή null). Παρόλα αυτά είναι καλή ιδέα να αρχικοποιείτε τα πεδία ακόμη και αν θέλετε τις εξ ορισμού τιμές, για να είναι σαφές ποιες είναι οι αρχικές τιμές. Επιπλέον δεν μπορείτε να χρησιμοποιήσετε ένα πεδίο αν δεν του έχετε αποδώσει κάποια τιμή.
- Μπορείτε να αρχικοποιήσετε τα πεδία και να αποδώσετε σε αυτά τις αρχικές επιθυμητές τιμές κατά την κατασκευή αντικειμένων της κλάσης με την χρήση ενός ή περισσότερων κατασκευαστών (constructors).

Κατασκευαστές

- Ένας κατασκευαστής είναι μια ειδική μέθοδος που εκτελείται όταν δημιουργείτε αντικείμενα μιας κλάσης. Πρέπει υποχρεωτικά να έχει το ίδιο όνομα με την κλάση, ενδέχεται να έχει παραμέτρους, αλλά δεν επιτρέπεται να επιστρέφει κάποια τιμή (ούτε void).
- Κάθε κλάση πρέπει να έχει τουλάχιστον έναν κατασκευαστή. Αν δεν δημιουργήσετε εσείς έναν, τότε θα δημιουργηθεί αυτόματα από τον μεταγλωττιστή ένας εξ ορισμού κατασκευαστής χωρίς παραμέτρους.
- Μπορείτε να γράψετε τον δικό σας εξ ορισμού κατασκευαστή (δηλ. έναν κατασκευαστή χωρίς παραμέτρους) πολύ εύκολα. Απλά προσθέστε μία δημόσια μέθοδο με το ίδιο όνομα με αυτό της κλάσης και χωρίς τύπο επιστροφής.
- Τυπικά στον κατασκευαστή αρχικοποιούμε τα πεδία στις επιθυμητές αρχικές τους τιμές και γενικά κάνουμε οτιδήποτε χρειάζεται να γίνει κατά την δημιουργία ενός νέου αντικειμένου της κλάσης.

Παράδειγμα κατασκευαστή

```
class Circle
{
    private int radius;
    public Circle() // default constructor
    {
        radius = 0;
    }
    public double Area()
    {
        return Math.PI * radius * radius;
    }
}
```

Μία καλή σύμβαση (δηλ. συνήθεια) στην C# είναι πως τα δημόσια πεδία και μέθοδοι ξεκινούν με κεφαλαίο γράμμα (π.χ. Area) ενώ τα ιδιωτικά πεδία και μέθοδοι ξεκινούν με μικρό γράμμα (π.χ. radius).

Παράδειγμα χρήσης αντικειμένου

```
class Circle
{
    int radius;
    double Area()
    {
        return Math.PI * radius * radius;
    }

    public static void Main(String[] args)
    {
        Circle c;
        c = new Circle();
        double areaOfCircle = c.Area();
        Console.WriteLine("Circle area is : {0}", areaOfCircle);
        Console.ReadKey();
    }
}
```

Υπερφόρτωση κατασκευαστών

- Σε μία κλάση μπορούμε να έχουμε πολλούς κατασκευαστές που διαφέρουν όμως ως προς τη λίστα παραμέτρων τους
- Για παράδειγμα μπορούμε να έχουμε ακόμη έναν κατασκευαστή για την κλάση `Circle` ο οποίος δέχεται ως παράμετρο έναν ακέραιο αριθμό για την αρχικοποίηση της ακτίνας του κύκλου
- Η δυνατότητα να έχουμε πολλές μεθόδους με το ίδιο όνομα σε μία κλάση οι οποίες όμως διαφέρουν ως προς την λίστα των παραμέτρων τους ονομάζεται *υπερφόρτωση* (overloading)
- Σημαντική επισήμανση: Στην C# αν γράψετε έναν κατασκευαστή με παραμέτρους τότε ο εξ ορισμού κατασκευαστής (χωρίς παραμέτρους) δεν παράγεται αυτόματα. Αν τον χρειάζεστε θα πρέπει να τον γράψετε.

```

class Circle
{
    int radius;
    public Circle() // default constructor
    {
        radius = 0;
    }
    public Circle(int initialRadius) // overloaded constructor
    {
        radius = initialRadius;
    }
    double Area()
    {
        return Math.PI * radius * radius;
    }
    public static void Main(String[] args)
    {
        Circle c;
        //Εδώ καλούμε τον υπερφορτωμένο κατασκευαστή
        c = new Circle(2);
        double areaOfCircle = c.Area();
        Console.WriteLine("Circle area is : {0}", areaOfCircle);
        Console.ReadKey();
    }
}

```

Μερικές (partial) κλάσεις

- Στην C# μπορείτε να «σπάσετε» τον ορισμό μιας κλάσης σε δύο ή και περισσότερα μέρη. Κάθε διαφορετικό μέρος της κλάσης δηλώνεται με τον ίδιο τρόπο που θα δηλώναμε την κλάση αν δεν την σπάζαμε σε μέρη, αλλά χρησιμοποιούμε σε κάθε μέρος της κλάσης την λέξη-κλειδί *partial*.
- Κατά την μεταγλώττιση του προγράμματος τα διάφορα μέρη μιας κλάσης συνενώνονται για να συνθέσουν ολόκληρη την κλάση.
- Στο Visual Studio 2010 αυτή η τεχνική χρησιμοποιείται κατά την σχεδίαση γραφικών διασυνδέσεων για να διαχωριστεί ο κώδικας που παράγεται αυτόματα από την γραφική σχεδίαση της διασύνδεσης χρήστη από τον κώδικα που γράφουν οι προγραμματιστές.

Παράδειγμα μερικής κλάσης #1

partial class Circle

```
{
    int radius;
    public Circle() // default constructor
    {
        radius = 0;
    }
    double Area()
    {
        return Math.PI * radius * radius;
    }
    public static void Main(String[] args)
    {
        Circle c;
        //Αυτός ο constructor βρίσκεται σε άλλο αρχείο
        c = new Circle(2);
        double areaOfCircle = c.Area();
        Console.WriteLine("Circle area is : {0}", areaOfCircle);
        Console.ReadKey();
    }
}
```

Παράδειγμα μερικής κλάσης #2

partial class Circle

```
{  
    public Circle(int initialRadius) // overloaded constructor  
    {  
        radius = initialRadius;  
    }  
}
```

Άσκηση

- Δημιουργείστε μία κλάση Point. Η Point θα έχει δύο ιδιωτικά πεδία (x και y) για τις συντεταγμένες του σημείου. Τα x και y είναι double για να μπορούν να πάρουν πραγματικές τιμές. Δημιουργείστε δύο κατασκευαστές, έναν εξ ορισμού που αρχικοποιεί τα x και y σε 0, και έναν που δέχεται δύο παραμέτρους και αρχικοποιεί τα x και y στις τιμές των παραμέτρων.

- Δημιουργείστε μία μέθοδο με την μορφή **public double DistanceTo(Point other)**

Η μέθοδος αυτή υπολογίζει και επιστρέφει την απόσταση του σημείου στο οποίο καλείται από τα σημείο που δίνεται ως παράμετρος. Αν συμβολίσουμε τα δύο σημεία ως (x1,y1) και (x2,y2) αντίστοιχα τότε η απόστασή τους είναι

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Για την τετραγωνική ρίζα χρησιμοποιείστε την Math.Sqrt

- Δημιουργείστε ένα πρόγραμμα Main που δημιουργεί 2 σημεία, ένα με τον εξ ορισμού κατασκευαστή και ένα με τον παραμετρικό κατασκευαστή, υπολογίζει την απόσταση του πρώτου από το δεύτερο και εμφανίζει αυτήν την απόσταση στην κονσόλα.

Static μέθοδοι και πεδία

- Υπάρχουν περιπτώσεις στις οποίες θα ήταν χρήσιμο να έχουμε κάποια δεδομένα τα οποία ανήκουν στη κλάση και όχι στα αντικείμενα της κλάσης. Αυτά τα πεδία τα δηλώνουμε ως static. Αντίστοιχα είναι επιθυμητό κάποιες φορές, οι μέθοδοι να μπορούν να κληθούν απλά χρησιμοποιώντας το όνομα της κλάσης. Έτσι αποφεύγουμε τη δημιουργία ενός αντικειμένου απλά και μόνο για να καλέσουμε τη μέθοδο.
- Αυτό βεβαίως έχει νόημα μόνο όταν η μέθοδος δεν χρησιμοποιεί δεδομένα του αντικειμένου. Έτσι λοιπόν μία μέθοδος που δηλώνεται ως static μπορεί να αναφερθεί μόνο σε δεδομένα τα οποία έχουν δηλωθεί ως static ή στις παραμέτρους της.
- Παράδειγμα static μεθόδων είδαμε ήδη για την κλάση Math. Για παράδειγμα η `Math.Sqrt(double x)`, επιστρέφει την τετραγωνική ρίζα της παραμέτρου της. Για να το κάνει αυτό δεν χρειάζεται πρόσβαση σε δεδομένα κάποιου αντικειμένου τύπου Math. Το μόνο που χρειάζεται είναι η παράμετρος.
- Μία static μέθοδος καλείται χρησιμοποιώντας το όνομα της κλάσης, ακολουθούμενο από μία τελεία, ακολουθούμενη από το όνομα της μεθόδου: π.χ. `Math.Sqrt`

Παράδειγμα static πεδίων & μεθόδων

```
class Person
```

```
{  
    private static int noOfPersons = 0;  
    public Person()  
    {  
        noOfPersons++;  
    }  
    public static int howMany()  
    {  
        return noOfPersons;  
    }  
    public static void Main()  
    {  
        Person p1 = new Person();  
        Person p2 = new Person();  
        Console.WriteLine("Δημιουργήθηκαν {0} Persons", Person.howMany());  
        Console.ReadKey();  
    }  
}
```

Τι θα εμφανίσει
αυτό το
πρόγραμμα;

ΣΤΑΤΙΚΑ ΣΤΑΘΕΡΑ ΠΕΔΙΑ

- Στην C# η χρήση της λέξης-κλειδί `const` στην δήλωση ενός πεδίου σημαίνει ότι το πεδίο αυτό θα έχει σταθερή τιμή μετά την αρχικοποίησή του και δεν μπορεί να αλλάξει αυτή η τιμή. Τα πεδία που δηλώνονται ως `const` είναι εξ ορισμού `static` παρότι δεν αναγράφουμε την λέξη-κλειδί `static`.
- Για παράδειγμα η δήλωση της σταθεράς `PI` στην κλάση `Math` μπορεί να γίνει ως εξής:
`public const double PI = 3.14159265358979323846;`
- Τα πεδία `const` μπορεί να είναι μόνο αριθμητικά (π.χ. `int` ή `double`), `string` ή `Enumerations` (για `Enumerations` θα μιλήσουμε σε επόμενο μάθημα).

ΣΤΑΤΙΚΕΣ ΚΛΑΣΕΙΣ

- Στην C# μπορείτε να δηλώσετε μία κλάση ως static.
- Μία στατική κλάση μπορεί να περιέχει αποκλειστικά και μόνο στατικά πεδία και μεθόδους και δεν μπορείτε να δημιουργήσετε αντικείμενά της (θα προκληθεί λάθος κατά την μεταγλώττιση αν το κάνετε)
- Ο λόγος που θα δηλώνατε μία κλάση ως static είναι για να ομαδοποιήσετε πολλές στατικές μεθόδους (π.χ. βοηθητικές μεθόδους σε σχέση με μία λειτουργία της εφαρμογής σας).
- Αν για παράδειγμα δηλώναμε εμείς την δική μας κλάση Math θα μπορούσαμε να την δηλώσουμε ως static αν δεν είχε κανένα πεδίο αντικειμένων παρά μόνο στατικές (βοηθητικές) μεθόδους. Σημειώστε πως η πραγματική κλάση Math στην C# δε είναι static γιατί περιέχει κάποιες μεθόδους σε επίπεδο αντικειμένου (όχι στατικές).

public static class Math

```
{  
    public static double Sin(double x) {...}  
    public static double Cos(double x) {...}  
    public static double Sqrt(double x) {...}  
    ...  
}
```

Ανώνυμες Κλάσεις

- Ανώνυμες κλάσεις είναι μη στατικές κλάσεις που δεν έχουν όνομα, μπορούν να έχουν μόνο αρχικοποιημένα δημόσια πεδία και δεν μπορούν να έχουν μεθόδους.
- Μία ανώνυμη κλάση χρησιμοποιείτε στο σημείο που δηλώνεται με την ανάθεσή της σε μία μεταβλητή. Επειδή δεν γνωρίζουμε τον τύπο της κλάσης αντί για τον τύπο της χρησιμοποιούμε την λέξη-κλειδί **var** ώστε η μεταβλητή να έχει τον υπονοούμενο τύπο της έκφρασης που καταχωρείται (δηλ. της ανώνυμης κλάσης σε αυτή τη περίπτωση).
- Για παράδειγμα η ακόλουθη δήλωση:

```
var myAnonymousObject =  
    new { Name = "John", Age = 44 };
```

- Δύο ανώνυμες κλάσεις θεωρείτε πως έχουν τον ίδιο τύπο αν έχουν τα ίδια πεδία, με τα ίδια ονόματα, με την ίδια σειρά, και με τον ίδιο τύπο.
- Για παράδειγμα το `anotherAnonymousObject` έχει τον ίδιο τύπο με το `myAnonymousObject`, στο παράδειγμα:

```
var anotherAnonymousObject =  
    new { Name = "Diana", Age = 45 };
```

Η ανάγκη για απόκρυψη των δεδομένων (data hiding principle)

- Όπως ήδη αναφέραμε τα πεδία μιας κλάσης είναι συνήθως ιδιωτικά. Αυτό πρέπει να ισχύει διότι η ορθότητα των δεδομένων μιας κλάσης θεωρείται αρμοδιότητα της κλάσης. Αν η κλάση επέτρεπε την δημόσια πρόσβαση σε αυτά τότε οι προγραμματιστές (κατά λάθος ή εσκεμμένα) θα μπορούσαν να βάλουν οποιεσδήποτε τιμές στα πεδία της κλάσης.
- Επιπλέον η εσωτερική αναπαράσταση των δεδομένων είναι επίσης αρμοδιότητα της κλάσης. Για παράδειγμα ένα πεδίο που εξωτερικά φαίνεται να έχει τον τύπο A, θα μπορούσε εσωτερικά να αποθηκεύεται με τον τύπο B (π.χ. μία ημερομηνία θα μπορούσε να αποθηκεύεται εσωτερικά ως η ημερολογιακή απόσταση από μία ημερομηνία βάσης – δηλ. ως ένας αριθμός long).
- Σκεφτείτε για παράδειγμα μία κλάση Person που ενσωματώνει ένα πεδίο για την ηλικία του Person, έστω age τύπου int. Παρότι ένας int μπορεί να πάρει αρκετά μεγάλες αριθμητικές τιμές, θα ήταν παράλογο να επιτρέψουμε την εισαγωγή μιας τιμής για την ηλικία πάνω από 130 ή την εισαγωγή μιας αρνητικής τιμής.

Σύμφωνα με την Wikipedia το ρεκόρ μακροβιότητας κατέχει η Γαλλίδα Jeanne Louise Calment η οποία απεβίωσε σε ηλικία 122 ετών και 164 ημερών το 1997. Είχε γνωρίσει προσωπικά τον Vincent van Gogh, και είδε να χτίζουν τον πύργο του Eiffel.

Οι μέθοδοι κλάσεων ως σημεία ελέγχου

- Για να αποτρέψουμε την λάθος εισαγωγή τιμών στα πεδία μιας κλάσης και γενικά για να τα ελέγξουμε, θα πρέπει να γράψουμε κάποιον σχετικό κώδικα ελέγχου των τιμών. Ο κώδικας αυτός μπορεί να γραφεί μόνο στις μεθόδους μιας κλάσης.
- Αυτό όμως δυστυχώς κάνει την πρόσβαση στα πεδία κάπως περίεργη, με την έννοια πως για να θέσουμε τις τιμές των πεδίων ή για να τις ανακτήσουμε θα πρέπει να καλέσουμε μεθόδους.
- Ας δούμε ένα παράδειγμα:

```
class Employee
```

```
{  
    private int age;  
    public Employee(int age)  
    {  
        if (checkAge(age))  
        {  
            this.age = age;  
        }  
        else  
        {  
            this.age = 0;  
        }  
    }  
    public void setAge(int age)  
    {  
        if (checkAge(age))  
        {  
            this.age = age;  
        }  
    }  
}
```

```
public int getAge()
```

```
{  
    return this.age;  
}  
private bool checkAge(int age)  
{  
    return (age >= 0 && age <= 130);  
}  
public static void Main()  
{  
    Employee e1 = new Employee(32);  
    Console.WriteLine("Η ηλικία του e1 είναι: {0}",  
        e1.getAge()); //ανάκτηση της ηλικίας του e1  
    e1.setAge(33); //αλλαγή της ηλικίας του e1  
    Console.WriteLine(  
        "Η ηλικία του e1 τώρα είναι: {0}",  
        e1.getAge()); //ανάκτηση της ηλικίας του e1  
    Console.ReadKey();  
}
```


Συζήτηση του παραδείγματος

- Παρότι το παράδειγμα δουλεύει η αλλαγή της τιμής του `age` πρέπει να γίνει μέσω της μεθόδου `setAge` (η οποία καλεί και την εσωτερική ιδιωτική μέθοδο `checkAge` για να ελέγξει αν η ηλικία είναι στα επιτρεπτά όρια) και η ανάκτηση της τιμής της γίνεται μέσω της κλήσης της μεθόδου `getAge`. Αυτό κάνει το πρόγραμμα κάπως δυσανάγνωστο.
- Θα ήταν προτιμότερο να μπορούμε να πούμε: `e1.Age=34` αντί για `e1.setAge(34)`; και `Console.WriteLine(e1.Age)` αντί για `Console.WriteLine(e1.getAge())`.
- Οι ιδιότητες (properties) είναι μία επινόηση της C# (δεν υπάρχουν στη Java) για να λυθεί αυτό το πρόβλημα. Παρέχουν πρόσβαση σαν να πρόκειται για δημόσια πεδία αλλά στην πραγματικότητα είναι μέθοδοι και επομένως μπορούμε να βάλουμε κώδικα ελέγχου γι' αυτές ως συνήθως.

Σύνταξη ιδιοτήτων

- Το συντακτικό για την δήλωση μιας ιδιότητας είναι το ακόλουθο:

Πρόσβαση Τύπος ΌνομαΙδιότητας

```
{  
    get  
    {  
        <ΕΝΤΟΛΕΣ ΠΟΥ ΕΚΤΕΛΟΥΝΤΑΙ ΟΤΑΝ ΑΝΑΚΤΟΥΜΕ ΤΗΝ ΙΔΙΟΤΗΤΑ>  
    }  
    set  
    {  
        <ΕΝΤΟΛΕΣ ΠΟΥ ΕΚΤΕΛΟΥΝΤΑΙ ΟΤΑΝ ΑΛΛΑΖΟΥΜΕ ΤΙΜΗ ΣΤΗΝ ΙΔΙΟΤΗΤΑ>  
    }  
}
```

- Ας δούμε το προηγούμενο παράδειγμα με την χρήση μιας δημόσιας ιδιότητας Age (προσέξτε την διαφορά στο κεφαλαίο A). Η δημόσια ιδιότητα λέγεται Age, το ιδιωτικό πεδίο λέγεται age. Φυσικά θα μπορούσαμε να χρησιμοποιήσουμε και οποιοδήποτε άλλο όνομα για την ιδιότητα.

```
class Employee1
```

```
{  
    private int age;  
    public Employee1(int age)  
    {  
        if (checkAge(age))  
        {  
            this.age = age;  
        }  
        else  
        {  
            this.age = 0;  
        }  
    }  
    public int Age  
    {  
        set  
        {  
            if (checkAge(age))  
            {  
                //προσέξτε την λέξη-κλειδί value  
                this.age = value;  
            }  
        }  
        get  
        {  
            return this.age;  
        }  
    }  
}
```

```
private bool checkAge(int age)  
{  
    return (age>=0 && age<=130);  
}  
  
public static void Main()  
{  
    Employee1 e1 = new Employee1(32);  
    Console.WriteLine("Η ηλικία του e1 είναι: {0}",  
        e1.Age); //ανάκτηση της ηλικίας του e1  
    e1.Age=33; //αλλαγή της ηλικίας του e1  
    Console.WriteLine(  
        "Η ηλικία του e1 τώρα είναι: {0}",  
        e1.Age); //ανάκτηση της ηλικίας του e1  
    Console.ReadKey();  
}
```

Περιορισμοί των ιδιοτήτων

- Μία ιδιότητα μπορεί να έχει μόνο δύο μεθόδους `get` και `set` χωρίς παραμέτρους. Δεν μπορεί να περιέχει άλλες μεθόδους. Μπορείτε αν θέλετε να παραλείψετε την `set` (οπότε η ιδιότητα είναι μόνο για ανάγνωση – `read only`) ή την `get` (οπότε η ιδιότητα είναι μόνο για εγγραφή – `write only`) αλλά όχι και τις δύο.
- Μία ιδιότητα δεν μπορεί να χρησιμοποιηθεί σαν *ref* ή *out* παράμετρος σε μία μέθοδο (θα μιλήσουμε για *ref* και *out* παραμέτρους σε επόμενο μάθημα).

Περίληψη #1

Δήλωση κλάσης

Γράψτε την λέξη κλειδί **class** ακολουθούμενη από το όνομα της κλάσης και δύο άγκιστρα. Τα πεδία και οι μέθοδοι της κλάσης αναγράφονται ανάμεσα στα άγκιστρα:

```
class Person  
{  
    <πεδία και μέθοδοι της κλάσης Person>  
}
```

Δήλωση κατασκευαστή

Γράψτε μία μέθοδο με το ίδιο όνομα με αυτό της κλάσης, χωρίς τύπο επιστροφής (ούτε void). Για παράδειγμα:

```
class Point  
{  
    public Point(int x, int y)  
    {  
        <εντολές του κατασκευαστή>  
    }  
}
```

Περίληψη #2

Κλήση κατασκευαστή	<p>Χρησιμοποιείτε την λέξη-κλειδί new για να δημιουργήσετε ένα νέο αντικείμενο καλώντας τον κατάλληλο κατασκευαστή.</p> <p>Για παράδειγμα:</p> <p>Point origin = new Point(0, 0);</p>
Δήλωση στατικής μεθόδου	<p>Χρησιμοποιείτε την λέξη-κλειδί static στην δήλωση της μεθόδου. Για παράδειγμα:</p> <pre>class Point { public static int ObjectCount() { <εντολές της static μεθόδου> } }</pre>

Περίληψη #3

Κλήση στατικής μεθόδου	<p>Γράψε το όνομα της κλάσης, ακολουθούμενο από μία τελεία, ακολουθούμενο από το όνομα της μεθόδου.</p> <p>Για παράδειγμα:</p> <pre>int pointsCreatedSoFar = Point.ObjectCount();</pre>
Δήλωση στατικού πεδίου	<p>Χρησιμοποιείστε την λέξη-κλειδί static στην δήλωση του πεδίου. Για παράδειγμα:</p> <pre>class Point { private static int objectCount; <άλλα πεδία και μέθοδοι της κλάσης> }</pre>
Δήλωση σταθερού (const) πεδίου	<p>Χρησιμοποιείστε την λέξη-κλειδί const στην δήλωση του πεδίου. Για παράδειγμα:</p> <pre>public const double PI = ...;</pre>

Περίληψη #4

Προσπέλαση στατικού πεδίου	<p>Γράψε το όνομα της κλάσης, ακολουθούμενο από μία τελεία, ακολουθούμενο από το όνομα του πεδίου.</p> <p>Για παράδειγμα:</p> <pre>double area = Math.PI * radius * radius;</pre>
Δήλωση ιδιότητας	<p>Μια ιδιότητα δηλώνετε ως εξής:</p> <pre>Πρόσβαση Τύπος Όνομαιδιότητας { get { ... } set { ... } }</pre>

Ερωτήσεις;

