Εισαγωγή σε αντικειμενοστραφή concepts

Και λίγη C#

Κλάσεις

- Κλάση: τύπος δεδομένων που αποτελεί συλλογή πεδίων, ορισμών συναρτήσεων/μεθόδων και ορισμών άλλων τύπων δεδομένων.
- Αντίστοιχο σκεπτικό με struct σε C, με τη διαφορά ότι μπορεί να περιέχει και συναρτήσεις καθώς και άλλους τύπους δεδομένων.
- Η δήλωση μιας κλάσης αποτελεί τον ορισμό του τύπου και τον τρόπο κατασκευής ενός αντικειμένου.
- Είναι σημαντικό να διαχωριστεί η έννοια του αντικειμένου από την έννοια της κλάσης. Ένα αντικείμενο έχει σαν τύπο μια κλάση, η οποία απλά προσδιορίζει από τι απαρτίζεται το αντικείμενο.

Κλάσεις

- Ο ορισμός μια κλάσης στη C# μπορεί να έχει την εξής μορφή:

```
class MyClass
{
        /*Pedia*/
}
```

```
class MyClass
{
        int myInt;
        float myFloat = 0.2f;
        string myString;

        void myFunction()
        {

        }
}
```

Ορατότητα & προσβασιμότητα

- Για λόγους αφαιρετικότητας αλλά και οργάνωσης του κώδικα, δίνεται η δυνατότητα να οριστεί το κατά πόσο κάποιο στοιχείο μιας κλάσης θα είναι ορατό από άλλες κλάσεις του ίδιου αρχείου ή και διαφορετικών αρχείων.
- Ο τρόπος χαρακτηρισμού του επιπέδου ορατότητας δίνεται από τις λέξεις-κλειδιά “public” και “private”.
- Ως public δηλώνονται τα στοιχεία τα οποία είναι δημόσια και έξω από την κλάση και συνεπώς μπορούν να προσπελαστούν ή και να μεταβληθούν από κάποια πηγή εκτός της κλάσης στην οποία ανήκουν.

Ορατότητα & προσβασιμότητα

- Ως private δηλώνονται τα στοιχεία τα οποία είναι ιδιωτικά για την ίδια την κλάση και δεν είναι καν ορατά για κάποια άλλη κλάση.
- Για παράδειγμα έστω οι εξής κλάσεις:

```
class MyClass
{
        public int myInt;
        private float myFloat = 0.2f;
        public string myString;

        public void myFunction()
        {
                myFloat++;
        }
}
```

```
class MyOtherClass
{
        void myOtherFunction()
        {
                MyClass myClassObject = new MyClass();
                myClassObject.myInt++;
                myClassObject.myFunction();
        }
}
```

Ορατότητα & προσβασιμότητα

- Στην κλάση MyClass, η συνάρτηση myFunction δεν έχει πρόβλημα να αλλάξει την τιμή του myFloat καθώς, αν και private, η μεταβολή γίνεται από μέθοδο της ίδιας κλάσης.
- Η μέθοδος myOtherFunction της κλάσης MyOtherClass μπορεί να μεταβάλλει την τιμή myInt του αντικειμένου myClassObject (το οποίο είναι τύπου MyClass) καθώς το συγκεκριμένο πεδίο έχει δηλωθεί ως public. Αντίστοιχα, μπορεί να καλεί τη συνάρτηση myFunction του ίδιου αντικειμένου καθώς και αυτή έχει οριστεί σαν public.
- Ωστόσο, δεν θα μπορούσε να δει ή να μεταβάλλει άμεσα την τιμή myFloat του αντικειμένου, καθώς το πεδίο έχει δηλωθεί ως private.

- Ας εστιάσουμε τη προσοχή μας στα στοιχεία αυτής της γραμμής από το προηγούμενο παράδειγμα:

```
MyClass myClassObject = new MyClass();
```

```
MyClass myClassObject = new MyClass();
```

- Το όνομα της κλάσης που αποτελεί τον τύπο του αντικειμένου που θέλουμε να φτιάξουμε.
- Το σκεπτικό είναι το ίδιο όπως όταν δηλώνουμε έναν ακέραιο: int i = 3;

```
MyClass myClassObject = new MyClass();
```

- Το όνομα του αντικειμένου τύπου MyClass το οποίο επιθυμούμε να κατασκευάσουμε.

```
MyClass myClassObject = new MyClass();
```

- Λέξη κλειδί που συναντάται και στην C++.
- Δηλώνει την δυναμική δέσμευση μνήμης για το αντικείμενο τύπου MyClass.

References

- Όπως και στη Java, στη C# κάθε πεδίο (εκτός από αυτά των primitive types: int, float, string, bool etc) είναι στην πραγματικότητα μια αναφορά (reference).
- Τα references έχουν αντίστοιχη έννοια με αυτή των δεικτών (pointers) σε γλώσσες όπως η C/C++.
- Η εντολή “MyClass myClassObject;” ΔΕΝ δημιουργεί ένα αντικείμενο τύπου MyClass αλλά μια ΑΝΑΦΟΡΑ σε αντικείμενο τύπου MyClass.
- Αν φτιαχτεί ένα πεδίο με τον τρόπο που αναφέρθηκε και δεν δοθεί τιμή στην αναφορά, κάθε προσπάθεια προσπέλασης θα δημιουργεί προβλήματα.

References

C++

```
void Function() {
        MyClass myClassObject;
        cout <<
        myClassObject.myInt <<
         endl;
}
```

Αυτός ο κώδικας θα εκτελεστεί κανονικά, άσχετα αν η εκτύπωση δείξει «σκουπίδια»

C#

```
void Function() {
        MyClass myClassObject;
        print(myClassObject.myInt);
}
```

Αυτός ο κώδικας δεν θα κάνει καν compile. Σε άλλες περιπτώσεις μπορεί να γίνεται compile αλλά να εμφανίζεται runtime error “Null reference exception”

```
MyClass myClassObject = new MyClass();
```

- Κλήση του constructor της κλάσης MyClass κατά τη δημιουργία του αντικειμένου.

Constructor

- Ο constructor είναι η συνάρτηση η οποία καλείται κατά τη δημιουργία ενός αντικειμένου της κλάσης. Χρησιμοποιείται κυρίως για την αρχικοποίηση των πεδίων του αντικειμένου.
- Το σώμα του constructor ορίζεται μέσα στην ίδια την κλάση ως μια συνάρτηση χωρίς τύπο επιστροφής με ίδιο όνομα με αυτό της κλάσης.
- Ο constructor μπορεί να παίρνει σαν ορίσματα τιμές τις οποίες μπορεί ενδεχομένως να αναθέτει στα πεδία του αντικειμένου της κλάσης.
- Εφόσον δεν οριστεί ρητά constructor μέσα στην κλάση (όπως στο παράδειγμα του MyClass), χρησιμοποιείται ένας default constructor.

Constructor

- Παραδείγματα constructor:

```
class MyClass {
    public int myInt;

    public MyClass() {
        myInt = 5;
    }
}
```

```
class MyClass {
    public int myInt;

    public MyClass(int myInt) {
        this.myInt = myInt;
    }
}
```

Σημειώνεται πως η λέξη κλειδί “this” αποτελεί αναφορά στο αντικείμενο της κλάσης που καλεί την τρέχουσα συνάρτηση. Χρησιμοποιείται συχνά για λόγους σαφήνειας, όπως στο 2ο παράδειγμα. Εκεί υπάρχει το πεδίο myInt και το όρισμα myInt του constructor, οπότε με το this.myInt προσδιορίζεται ρητά πως αναφερόμαστε στο πεδίο του αντικειμένου.

Static στοιχεία

- Εντός των κλάσεων μπορεί να υπάρχουν στοιχεία τα οποία θα ήταν χρήσιμο να σχετίζονται με την έννοια της κλάσης και όχι απλά με κάθε ένα αντικείμενο του τύπου της κλάσης.
- Ένα static πεδίο είναι ορατό από κάθε μέθοδο της κλάσης και εφόσον είναι public θα μπορεί να προσπελαστεί από οποιαδήποτε κλάση ως εξής: ΌνομαΚλάσης.ΌνομαStaticΠεδίου
- Συναρτήσεις της κλάσης μπορούν επίσης να είναι static και μπορούν να προσπελαστούν με αντίστοιχο τρόπο. Ωστόσο είναι σημαντικό να αναφερθεί πως μια static συνάρτηση μιας κλάσης δεν μπορεί να προσπελάσει μη-static πεδία της κλάσης. (Γιατί;)

Static στοιχεία

- Παραδείγματα ορισμού και χρήσης static πεδίων:

```
class Car
{
	public static int counter = 0;

	public Car()
	{
		counter++;
	}
}
```

```
class MyOtherClass
{
	void myOtherFunction()
	{
		Car car1 = new Car();
		Car.counter++;
	}
}
```

Κληρονομικότητα

- Ίσως ένα από τα πιο βασικά χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού, η κληρονομικότητα είναι η ιδιότητα που έχουν οι κλάσεις να οργανώνονται σε μια ιεραρχία γονιού-παιδιού.
- Μια κλάση-παιδί θα έχει όλα τα πεδία και τις μεθόδους που έχει η αντίστοιχη κλάση-γονιός, εφόσον δεν είναι private.
- Επιπλέον στοιχεία μπορούν έπειτα να προστεθούν σε μια κλάση-παιδί κάνοντάς την έτσι πιο ειδική από τη κλάση-γονιό της.

Κληρονομικότητα

- Παράδειγμα υλοποίησης κληρονομικότητας:

```
class Computer
{
        public int RAM;
        public string CPU;

        public Computer(int ram, string cpu)
        {
                RAM = ram;
                CPU = cpu;
        }
}
```

Κλάση-γονιός

```
class Smartphone : Computer
{
        public string number;
        public float battery_life;

        public Smartphone(int ram, string cpu, string number, float battery_life) :
        base(ram,cpu)
        {
                this.number = number;
                this.battery_life = battery_life;
        }
}
```

Κλάση-παιδί

Κληρονομικότητα

- Ως προς το παραπάνω παράδειγμα, σημειώνεται πως η κλάση Smartphone έχει επίσης τα public πεδία της κλάσης Computer και πως στον constructor της καλεί με τη χρήση του “base” τον constructor της κλάσης Computer, καθώς καλείται και αυτός λόγω κληρονομικότητας.
- Σημειώνεται πως σε ό,τι αφορά την κληρονομικότητα προστίθεται ένα ακόμα επίπεδο ορατότητας που προσδιορίζεται με τη λέξη-κλειδί “protected”. Όσα πεδία δηλωθούν ως protected σε μια κλάση-γονιό θα περαστούν μεν στις κλάσεις παιδιά, αλλά δεν θα είναι δημόσια για άλλες κλάσεις.

Πολυμορφισμός

- Άμεσα συνδεδεμένος με την έννοια της κληρονομικότητας, ο πολυμορφισμός είναι η ιδιότητα των αντικειμένων που έχουν σαν τύπο μια κλάση-παιδί να υιοθετούν ταυτόχρονα τον τύπο της κλάσης γονιού.
- Εφόσον απευθυνθούμε σε ένα αντικείμενο με τύπο την κλάση-παιδί ως αντικείμενο με τύπο την κλάση-γονιό, θα μπορούμε να προσπελάσουμε μόνο τα πεδία της κλάσης γονιού.

Πολυμορφισμός

- Για παράδειγμα, έστω οι εξής κλάσεις:

```
class A {
    public int myInt;
}
```

```
class B : A {
    public float myFloat;
}
```

Θα έχουμε τη δυνατότητα σε κάποιο σημείο του προγράμματος να γράψουμε το εξής:

```
A aObject = new B();
print(aObject.myInt);
```

Καθώς, λόγω κληρονομικότητας, ένα αντικείμενο της κλάσης Β είναι ταυτόχρονα και αντικείμενο της κλάσης Α. Ωστόσο, δεν θα μπορούσαμε να προσπελάσουμε το πεδίο myFloat καθώς το aObject είναι αναφορά σε αντικείμενο τύπου Α και όχι τύπου Β.

Shadowing & overriding

- Έστω πως οι κλάσεις του προηγούμενου παραδείγματος είχαν την εξής μορφή:

```
class A {
    public int myInt;
}
```

```
class B : A {
    public float myInt;
}
```

- Στην περίπτωση αυτή το πεδίο myInt της κλάσης B «κρύβει» (hides/shadows) το πεδίο myInt της κλάσης A και συνεπώς, σε ένα αντικείμενο τύπου B, το πεδίο myInt θα απευθύνεται μόνο στην float εκδοχή.
- Για να γίνει σαφές πως το shadowing ήταν σκόπιμο, χρησιμοποιείται στη C# η λέξη-κλειδί new ως εξής:

```
class B : A {
    new public float myInt;
}
```

Shadowing & overriding

- Μια συνάρτηση μιας κλάσης-γονιού μπορεί να δηλωθεί ως virtual, στην οποία περίπτωση η κλάση-παιδί μπορεί να έχει μια συνάρτηση με το ίδιο όνομα ως override συνάρτηση.
- Ο τρόπος δήλωσης μιας συνάρτησης ως override γίνεται με τη λέξη-κλειδί “override” και τα αποτελέσματα είναι εμφανή σε περίπτωση πολυμορφισμού.
- Ένα παράδειγμα θα εκφράσει καλύτερα την ιδιότητα μιας override συνάρτησης.

Shadowing & overriding

• Έστω οι εξής κλάσεις:

```
class A {                          class B : A {
    public void Func() {               public void Func() {
        print(“A”);                        print(“B”);
    }                                          }
}                                          }
```

Σε αυτήν την περίπτωση, η συνάρτηση Func της κλάσης Β κάνει shadow την συνάρτηση Func της κλάσης Α.

Shadowing & overriding

- Συνεπώς ο εξής κώδικας:

```
…
A aObject = new B();
aObject.Func();
```

Θα έχει ως αποτέλεσμα την εκτύπωση του γράμματος “A”, καθώς καλείται η Func της κλάσης A, αφού το aObject είναι αναφορά σε αντικείμενο τύπου A.

Shadowing & overriding

- Ωστόσο, αν οι κλάσεις ήταν έτσι:

```
class A {
    public virtual void Func() {
        print("A");
    }
}
```

```
class B : A {
    public override void Func() {
        print("B");
    }
}
```

Ο ίδιος κώδικας θα εμφάνιζε το γράμμα "B", καθώς παρακάμπτεται η συνάρτηση Func της κλάσης A.

Shadowing & overriding

- Για διαφοροποίηση των δύο:

```
class A
{
   public int Foo(){ return 5;}
   public virtual int Bar(){return 5;}
}
class B : A
{
   public new int Foo() { return 1;}
   public override int Bar() {return 1;}
}
```

- Η περίπτωση της συνάρτησης Foo είναι shadowing ενώ της Bar είναι overriding.

Function overloading

- Σε αντικειμενοστραφείς γλώσσες δίνεται η δυνατότητα function overloading, η οποία αφορά στον ορισμό πολλαπλών συναρτήσεων με το ίδιο όνομα αλλά με διαφορετικό πλήθος/τύπο ορισμάτων.
- Πρόκειται για ένα πολύ χρήσιμο χαρακτηριστικό καθώς μειώνει τη πολυπλοκότητα του κώδικα και δεν χρειάζεται να φτιάχνονται συναρτήσεις με διαφορετικά ονόματα εφόσον κάνουν την ίδια λειτουργία με διαφορετικά ορίσματα.
- Σημειώνεται πως η διαφοροποίηση μπορεί να γίνει μόνο στα ορίσματα, όχι στον τύπο επιστροφής.

Function overloading

- Παραδείγμα:

```
class MyClass {
    public string myInt;

    public void foo() { }
    public void foo(int i) { }
    public void foo(int i, int j) { }
    public void foo(float i, int j) { }
    public void foo(int i, float j) { }
}
```

Σημειώνεται πως κάθε συνάρτηση μπορεί να έχει τελείως διαφορετικό σώμα.

Generic methods

- Σημαντική δυνατότητα σε αντικειμονστραφείς γλώσσες είναι η χρήση generic κλάσεων και συναρτήσεων.
- Το σκεπτικό πίσω από τη χρήση generic κλάσεων/συναρτήσεων είναι πως κάποιες πράξεις πάνω σε πεδία μπορούν να γίνουν ανεξάρτητα από τον τύπο τους.
- Έστω, για παράδειγμα, η κλασική συνάρτηση swap, που παίρνει 2 ορίσματα και ανταλλάσσει τις τιμές τους. Με function overloading δεν θα ήταν καθόλου πρακτικό να καλύπτονται όλοι οι τύποι, καθώς θα χρειαζόταν πολύ χώρο και χρόνο για να φτιαχτούν όλες οι συναρτήσεις και θα έπρεπε γίνονται αναβαθμίσεις με κάθε καινούρια κλάση.

Generic methods

- Χρησιμοποιώντας την ιδιότητα των generic τύπων, αρκεί μόνο μία μέθοδος:

```
void Swap<T>(ref T lhs, ref T rhs)
{
    T temp;
    temp = lhs;
    lhs = rhs;
    rhs = temp;
}
```

- Εφόσον τα ορίσματα έχουν ορίσει την πράξη της ανάθεσης (=), αυτή η μέθοδος μπορεί να πάρει οποιοδήποτε τύπο ορισμάτων.
- Σημειώνεται πως η λέξη κλειδί “ref” στα ορίσματα δηλώνει πως παίρνει τα ορίσματα ως αναφορές, και συνεπώς οι αλλαγές σε αυτά μένουν και εκτός του σώματος της συνάρτησης.

Namespaces

- Namespace: πακέτο κλάσεων που χρησιμοποιείται για οργάνωση του κώδικα και για δυνατότητα ενός ακόμα επιπέδου αφαίρεσης.
- Ο ορισμός ενός namespace γίνεται περικλείοντας τις επιθυμητές κλάσεις σε ένα block ως εξής:

```
namespace MyNamespace
{
        //Classes…
}
```

Namespaces

- Για επιπλέον οργάνωση μπορεί να δοθεί ιεραρχία στα namespaces με την εξής μορφή:

```
namespace MyNamespace.FirstNamespace { …}
namespace MyNamespace.SecondNamespace {…}
```

Namespaces

- Παράδειγμα:

```
namespace myNamespace
{
        class MyClass
        {
                public void myFunction() { }
        }
}
```

- Η χρηση κλασεων απο συγκεκριμένο namespace γίνεται ως εξής:

```
using myNamespace;

    class MyOtherClass
    {
            void myOtherFunction()
            {
                    MyClass myClassObject = new MyClass();
                    myClassObject.myFunction();
            }
    }
```

ή

```
class MyOtherClass
    {
            void myOtherFunction()
            {
                    myNamespace.MyClass myClassObject = new myNamespace.MyClass();
                    myClassObject.myFunction();
            }
    }
```