Examen MVP

|  |
| --- |
| **Cap1 – Platforma Microsoft .NET** |

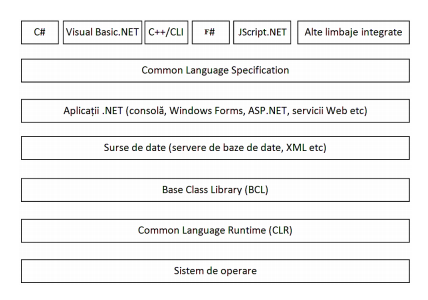
.NET contine:

* Biblioteca de clase
* CLR
* ASP .NET

Common Language Runtime(CLR) – mediul de rulare in .NET

* masina virtuala care pune la dispozitie servicii de Securitate, management al memoriei, manipulare de exceptii

Arhitectura platformei Microsoft .NET



Componente ale lui .NET Framework

* Common Intermidiate Language
  + constituie propria abstractizare de limbaj
  + Are trasaturi obiect orientate precum abstractizarea datelor, mostenirea, polimorfismul, chiar si exceptii si evenimente
  + Permite rularea aplicatiilor independent de platforma
* Common Language Specification
  + Contine specificatii de reguli necesare pentru integritatea limbajelor
  + Defineste un set de reguli pentru compilatoarele .NET
* Common Language Runtime
  + Cea mai importanta componenta a lui .NET
  + Responsabil cu managementul si executia codului scris in limbaje .NET
  + Instantiaza obiectele
  + Face verificari de Securitate
  + Depune obiectele in memorie
  + Disponibilizeaza memoria prin garbage collection
* Common Type System
  + Orice limbaj trebuie sa recunoasca si sa poata manipula niste tipuri comune
  + Exemple:
    - Tipuri valoare
      * Tipuri primitive
      * Structurile
      * Enumerarile
      * !!! Variabilele locale de tip valoare sunt alocate pe stiva
    - Tipuri referinta
      * Se folosesc daca variabilele de un anumit tip cer resurse de memorie semnificative
      * Contin adrese de memorie heap si pot fi null
      * Transferul de parametri se face rapid, dar referintele induc un cost suplimentar datorita garbage collection
    - Boxing si unboxing
      * Boxing – tipurile valoare se stocheaza pe stiva, dar pot fi convertite oricand intr-un tip referinta memorat in heap
      * Unboxing – obiectul se poate converti in tipul valoare echivalent
    - Clase, proprietati, indexatori
    - Interfete – o clasa care implementeaza o interfata trebuie sa implementeze toate metodele acelei interfete(daca se vrea a fi clasa instantiata)
    - Delegati
      * Inspirati de pointerii din C(eleganta mecanismului de callback)
      * Reprezinta versiunea “sigura” a pointerilor catre functii din C/C++
      * Mecanismul prin care se trateaza evenimentele

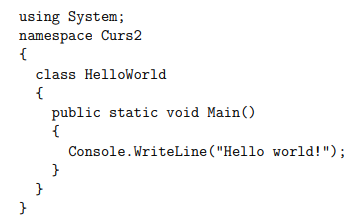
Componente importante ale CLR

1. Metadate
   1. date despre date
   2. detalii destinate a fi citite si folosite de catre platforma
   3. contin o declaratie a fiecarui tip si cate o declaratie pentru fiecare metoda, camp, proprietate, eveniment
2. Assemblies
   1. Reprezinta un bloc functional al unei aplicatii .NET
   2. Formeaza unitatea fundamentala de distribuire, versionare, reutilizare si permisiuni de securitate
   3. La runtime, un tip de date exista in interiorul unui assembly(si nu poate exista in exteriorul acestuia)
   4. Contine metadate care sunt folosite de catre CLR
   5. Scopul este sa se asigure dezvoltarea softului in mod “plug-and-play”
   6. Manifestul
      1. reprezinta metadate despre assembly-ul care gazduieste tipurile de date
      2. contine numele assembly-ului, numarul de versiune, referiri la alte assemblies, o lista a tipurilor in assembly, permisiuni de securitate si altele
3. Assembly cache
   1. Este un director aflat in mod normal in directorul Assembly
   2. Cand un assembly este instalat pe o masina, el va fi adaugat in assembly cache
   3. Exista versiuni multiple ale aceluiasi assembly
   4. Mod de rezolvare a problemei “dll hell”
4. Garbage collection
   1. Este mecanismul care se declanseaza atunci cand alocatorul de memorie raspunde negativ la o cerere de alocare de memorie
   2. CLR este cel care se ocupa de apelarea mecanismului de garbage collection

Trasaturi ale platformei .NET

* Dezvoltarea multilimbaj
* Independenta de procesor si platforma
* Managementul automat al memoriei
* Suportul pentru versionare
* Sprijinirea standardelor deschise
* Distribuire usoara
* Arhitectura distribuita
* Interoperabilitate cu codul “unmanaged”
* Securitate

|  |
| --- |
| **Cap2 – Vedere generala asupra limbajului C#. Tipuri predefinite. Tablouri. Siruri de caractere** |

****

Prima linie **using System**

* este o directiva care specifica faptul ca se vor folosi tipuri de date(de exemplu clase) incluse in spatial de nume **System**

**Spatiu de nume** = este o colectie de tipuri sau de alte spatii de nume care pot fi folosite intr-un program

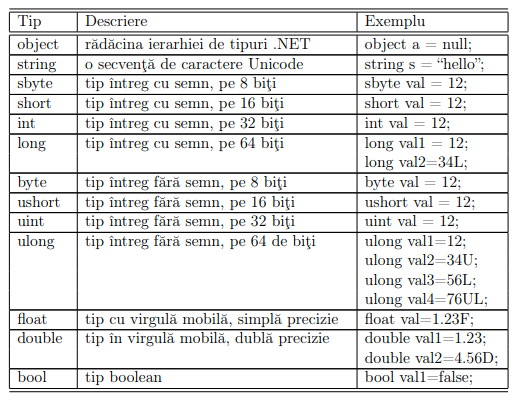
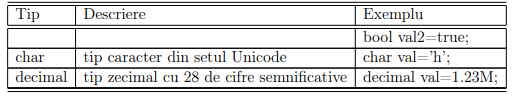
Conventii:

* conventie Pascal: prima litera a fiecarui cuvant se scrie ca litera mare: LoadData, SaveLogFile
  + este folosita pentru tot ce este vizibil(public), precum nume de clase, metode si proprietati.
* conventie de tip “camila”: la fel ca precedenta, dar primul character al primului cuvant este scris cu litera mica: userIdentifier, firstName
  + se foloseste pentru parametrii metodelor si numele campurilor
* conventie toate literele mari: System.IO, Math.PI, System.Web.UI
  + se foloseste pentru toti identificatorii de doua sau mai putine litere

**Tipuri de date**

Toate tipurile de date sunt derivate din tipul System.Object

* Tipuri valoare
  + Sunt alocate pe stiva sau inline intr-o structura
  + Exemple:
    - Tipuri simple(char, int, float)
    - Tipuri enumerare
    - Tipuri structura
* Tipuri referinta
  + Au proprietatea ca variabilele de acest tip stocheaza referinte catre obiectele continute
  + Tipuri clasa
  + Tipuri interfata
  + Tipuri delegat
  + Tipuri tablou
* **Tipuri predefinite**
* string – folosit pentru manipularea sirurilor de caractere codificate Unicode; continutul obiectelor de tip string nu se poate modifica
* bool – folosit pentru a reprezenta valorile logice true si false
* char – folosit pentru a reprezenta caractere Unicode
* decimal – folosit pentru calcule in care erorile determinate de reprezentarea in virgula mobile sunt inacceptabile(pune la dispozitie 28 de cifre zecimale semnificative)

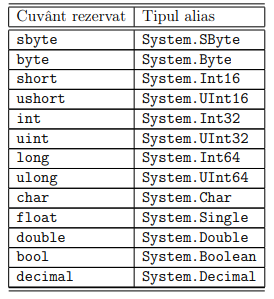
* **Tipuri valoare**

Toate tipurile valoare deriveaza din clasa System.ValueType, care la randul ei este derivata din clasa object(alias System.Object). Nu este posibil ca dintr-un tip valoare sa se deriveze.

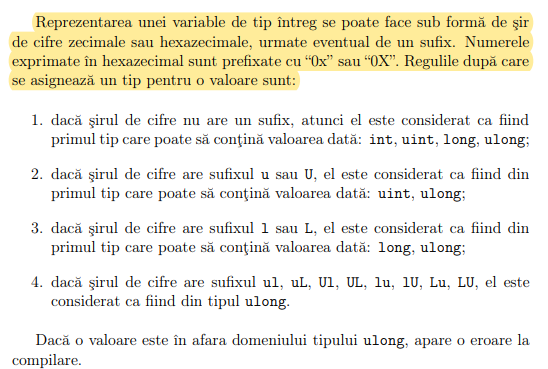
* Tipuri intregi: sbyte, byte, short, ushort, int, uint, long, ulong, char
* Tipuri in virgula mobila: float, double
* Tipuri enumerare: definite de utilizator

**Atribuirea** pentru un astfel de tip inseamna COPIEREA valorii dintr-o parte in alta

* Tipuri structura
  + Este un tip valoare care poate sa contina declaratii de constante, metode, proprietati, indexatori, operatori, constructori sau tipuri imbricate
* Tipuri simple
  + Sunt identificate prin cuvinte rezervate, aceste fiind doar alias-uri

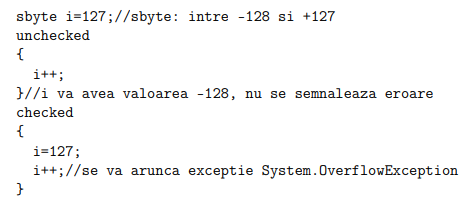


* Tipuri intregi
  + sbyte – tip cu semn pe 8 biti, intre -128 la 127
  + byte – tip fara semn pe 8 biti, intre 0 si 255
  + short – tip cu semn pe 16 biti, intre -32768 si 32767
  + ushort – tip fara semn pe 16 biti, intre 0 si 65535
  + int – tip cu semn pe 32 biti, intre -2^31 si 2^31 – 1
  + uint – tip fara semn pe 32 biti intre 0 si 2^32 – 1
  + long – tip cu semn pe 64 biti, intre -2^63 si 2^63 – 1
  + ulong – tip fara semn pe 64 biti intre 0 si 2^64 – 1
  + char – tip fara semn pe 16 biti intre 0 si 65535



Contextul CHECKED – o eroare de depasire duce la aruncarea unei exceptii de tip System.OverflowException

Contextul UNCHECKED – eroarea de depasire este ignorata, iar bitii semnificativi care nu mai incap in reprezentare sunt eliminate



!!! Daca nu se specifica nimic, atunci se considera implicit **unchecked**

* Tipuri in virgula mobile – float(32 biti) si double(64 biti)
  + Tipul float – poate reprezenta valori cuprinse intre 1.5x10-45 si 3.4 x 1038 cu precizie de 7 cifre
  + Tipul double – poate reprezenta valori cu modul cuprins intre 5.0 x 10-324 si 1.7 x 10308 cu o precizie de 15-16 cifre

Operatiile cu floating point nu duc la aparitia de exceptii, dar pot duce la operatii invalide, la valori 0, infinit sau NaN.

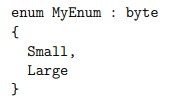
Literalii folositi: F, f, D, d

Daca niciun sufix de tip real nu e specificat, atunci literalul dat este de tip double. Sufixul f sau F specifica tip float, d sau D tipul double.

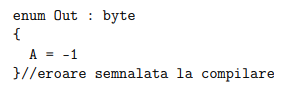
EROARE DE COMPILARE daca nu poate fi reprezentat tipul precizat

* Tipul decimal
  + - Este un tip de date reprezentat pe 128 de biti
    - Gandit pentru calcule financiare sau care necesita precizie mai mare
    - Reprezinta valori aflate in intervalul 1.0 x 10-28 si 7.9 x 1028 cu 28 de cifre semnificative
    - Nu poate reprezenta zero cu semn, infinit, sau NaN
    - Daca in urma operatiilor un numar este prea mic, atunci el e transformat in 0, iar daca e prea mare, rezulta exceptie.
    - Fata de tipurile in virgula mobile are precizie mai mare, dar domeniu mai mic
    - Nu se fac conversii implicite intre niciun tip in virgula mobile si decimal
    - Nu e posibila mixarea variabilelor de acest tip intr-o expresie, fara conversii explicite
    - Literalii folositi: m sau M
    - EROARE DE COMPILARE daca nu poate fi reprezentata prin tipul decimal
* Tipul bool
  + - este folosit pentru reprezentarea valorilor de adevarat si fals
    - Literalii folositi: true si false
    - Nu exista conversii implicite intre bool si niciun alt tip
    - Se reprezinta pe un octet
* **Tipul enumerare**
* este un tip valoare, construit pentru a permite declararea constantelor inrudite, intr-o maniera clara si sigura d.p.d.v. al tipului
* ca mod de scriere se sugereaza folosirea conventiei Pascal atat pentru numele tipului cat si pentru numele valorilor
* Nu pot fi declarate abstracte si nu pot fi derivate
* Sunt derivate automat din System.Enum care e derivate din System.ValueType
* Are un tip reprezentare, implicit int
* Tipurile de reprezentare pot fi: byte, sbyte, short, ushort, int, uint, long, ulong(alta varianta e rejectata la compilare)

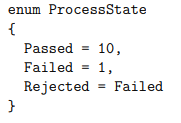
Exemplu:

****

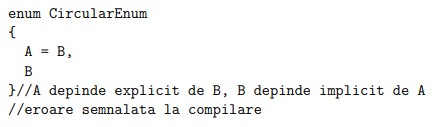
* Reguli:
  + valorile specificate ca initializatori trebuie sa fie compatibile cu tipul de reprezentare a enumerarii, altfel apare o eroare la compilare:

****

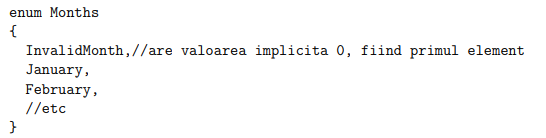
* + mai multi membri pot avea aceeasi valoare (manevra dictata de semantica tipului construit):



* + daca pentru un membru nu este data o valoare, acesta va lua valoarea membrului precedent + 1 (cu exceptia primului membru – vezi mai sus)
  + nu se permit referinte circulare



* + este recomandat ca orice tip enumerare sa contina un membru cu valoarea 0, pentru ca ın anumite contexte valoarea implicita pentru o entitate enumerare este 0, ceea ce poate duce la inconsistente¸ si bug-uri greu de depanat:



Tipurile enumerare pot fi convertite folosind cast

Valoarea 0 poate fi convertita catre un enum fara conversie explicita

**Tablouri**

* Fiecare tablou este un obiect, derivat din clasa abstracta System.Array
* Tablouri unidimensionale
* Tablouri multidimensionale: rectangulare si neregulate
  + Tablouri rectangulare:



* + - Au proprietatea ca numarul de elemente pentru o anumita dimensiune este pastrat constant
    - Au forma (hiper)dreptunghiulara.
    - Se pot folosi matrice cu mai mult de 2 dimensiuni(tensori)
    - GetLength – metoda care returneaza numarul de elemente aflate pe dimensiunea numarului d
    - Proprietatea Rank – determina numarul de dimensiuni pentru un tablou rectangular la run-time
  + Tablouri neregulate



* + - Reprezinta un tablou de tablouri
    - Este un vector care contine referinte, deci este unidimensional

**Siruri de caractere**

* Tipul de date este clasa System.String(alias string)
* Obiectele de acest tip sunt imuabile(caracterele nu se pot schimba, dar pe baza unui sir se poate obtine un alt sir)
* Sirurile pot fi de 2 feluri:
  + Siruri regulate
    - Sunt demarcate prin ghilimele si necesita secvente escape pentru reprezentarea caracterelor escape
  + Siruri de tip “verbatim”
    - Folosite in situatia in care se utilizeaza masiv secvente escape
    - Literal: “@”
    - Folosite pentru a face referiri la fisiere, chei de registri, sau expresii regulate
* Operatorii “==” si “!=”: doua siruri de caractere se considera egale daca sunt fie amandoua null, fie au aceeasi lungime si caracterele de pe aceleasi pozitii coincide
* Metode pentru
  + Comparare: Compare, CompareOrdinal, CompareTo
  + Cautare: EndsWith, StartsWith, IndexOf, LastIndexOf
  + Modificare: Concat, CopyTo, Insert, Join, PadLeft, PadRight, Remove, Replace, Split, ToLower, ToUpper

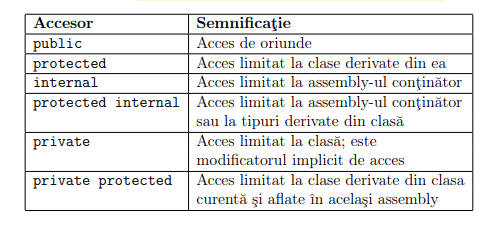
**Expresii regulate**

* Reprezinta o metoda facila de a opera cautari/inlocuiri pe text

|  |
| --- |
| **Cap3 – Clase-generalitati. Instructiuni. Spatii de nume** |

**Clase – vedere generala**

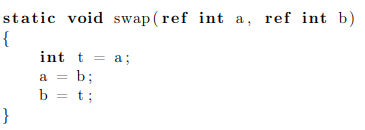
* Reprezinta tipuri referinta
* Poate sa mosteneasca direct o singura clasa
* Poate implementa mai multe interfete
* Pot contine:
  + Constante
  + Campuri
  + Metode
  + Proprietati
  + Evenimente
  + Indexatori
  + Operatori
  + Constructori de instanta
  + Destructori
  + Constructori de clasa
  + Tipuri imbricate
* Modificatorii de acces:



* Constanta
  + este un membru al unei clase care reprezinta o valoare nemodificabila, care poate fi evaluate la compilare
  + poate depinde de alte constante atata timp cat nu se creeaza dependente circulare
  + sunt considerate automat membri static, dar e **interzis** sa se foloseasca specificatorul **static** in fata lor
  + pot fi accesate exclusive prin intermediul numelui de clasa(MyClass.MyConst)
* Campul
  + Este un membru asociat fiecarui obiect
  + Stocheaza o valoare care contribuie la starea obiectului
* Metoda
  + Este un membru care implementeaza un calcul sau o actiune care poate fi efectuata asupra unui obiect sau asupra unei clase.
* Proprietatea
  + Este un membru care da acces la o caracteristica a unui obiect sau unei clase
* Evenimentul
  + Este un membru care permite unei clase sau unui obiect sa puna la dispozitia altora notificari asupra evenimentelor
  + Tipul trebuie sa fie un tip delegat
  + O instanta a unui tip delegate incapsuleaza una sau mai multe entitati apelabile
* Operatorul
  + Este un membru care defineste semnificatia(supraincarcarea) unui operator
  + Se pot supraincarca operatori binari, unari si de conversie
* Indexatorul
  + Este un membru care permite unui obiect sa fie indexat in acelasi mod ca un tablou
* Constructorii instanta
  + Sunt membri care implementeaza actiuni cerute pentru initializarea fiecarui obiect
* Destructorul
  + Este un membru special care implementeaza actiunile cerute pentru a dealoca resursele(altele decat de tip memorie) alocate de un obiect
  + Nu are parametri
  + Nu poate avea modificatori de acces
  + Nu poate fi apelat explicit
  + E apelat automat de catre garbage collector
* Constructorul static
  + Este un membru care implementeaza actiuni necesare pentru a initializa o clasa, mai exact membrii statici ai unei clase
  + Nu poate avea parametri
  + Nu poate avea modificatori de acces
  + Nu e apelat explicit
  + E apelat de catre sistem
* Mostenirea
  + Este de tip simplu – o clasa poate sa mosteneasca direct o singura clasa, radacina ierarhiei este clasa object(alias pentru System.Object)

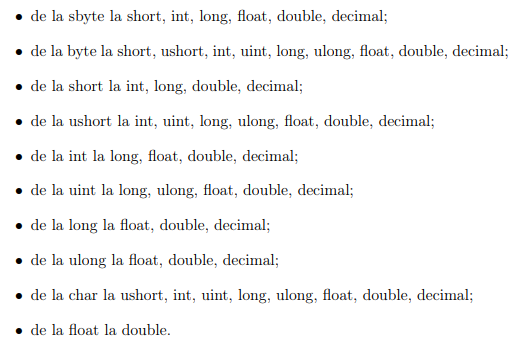
**Transmiterea de parametri**

* In mod implicit prin valoare
* Pentru variabilele de tip referinta, pe stiva se depune o copie a obiectului, adica adresa de memorie la care e stocat obiectul
* Un parametru referinta(ref) este folosit tocmai pentru a rezolva problema transmiterii prin valoare, folosind referinta(adica alias)
* Pentru a transmite un parametru prin referinta se prefixeaza cu cuvantul cheie ref la apel si la declarare de metoda

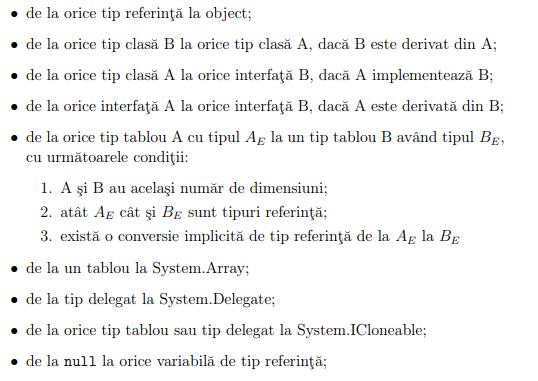


**Conversii**

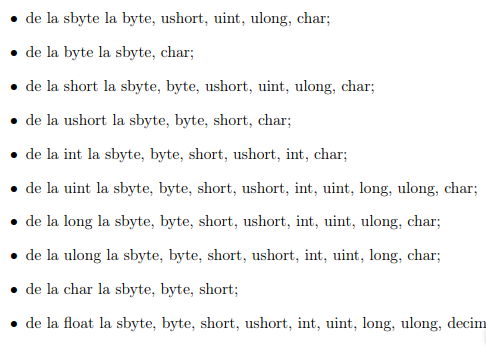
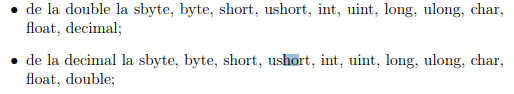
* Permite ca o expresie de un anumit tip sa fie tratata ca fiind de alt tip
* Pot fi implicite sau explicite
* Conversii implicite
  + Conversii identitate
    - Converteste de la un tip oarecare la acelasi tip
  + Conversii numerice implicite



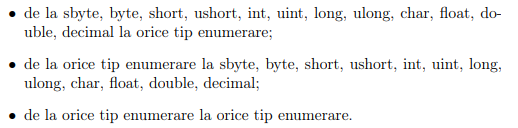
* + Conversii implicite de tip enumerare
    - Permite ca literalul 0 sa fie convertit la orice tip de enumerare
  + Conversii implicite de referinte

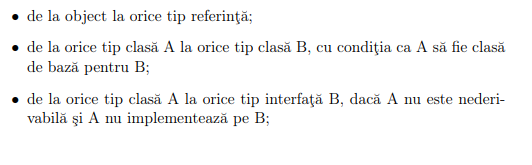


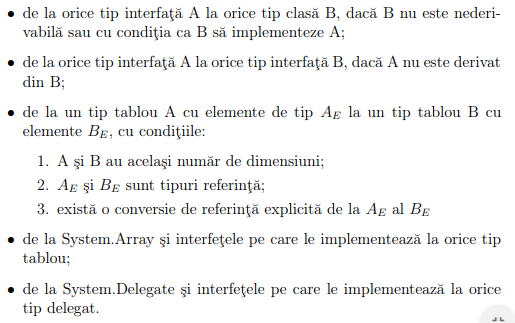
* + Boxing
    - Permite unui tip valoare sa fie implicit convertit catre un alt tip de date, aflat deasupra in ierarhie
  + Conversiile implicite ale expresiilor constante
  + Conversii implicite definite de utilizator
* Conversii explicite
  + Toate conversiile implicite
  + Conversiile numerice explicite

* + Conversiile explicite de enumerari



* + Conversiile explicite de referinte



* + Unboxing
    - Permite o conversie explicita de la object sau System.ValueType la orice tip valoare, sau de la orice tip interfata la orice tip valoare care implementeaza tipul interfata
  + Conversii explicite definite de utilizator

**Declaratii de variabile si constante**

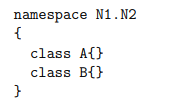
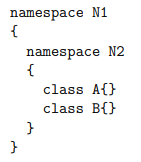
* Variabilele si constantele trebuie declarate in C# cu tip si nume
* O variabila trebuie sa aiba valoarea asignata definite inainte ca valoarea ei sa fie utilizata in cazul in care e utilizata in interiorul unei metode
* Este o eroare ca intr-un sub-bloc sa se declare o variabila cu acelasi nume ca in blocul continator

**Spatii de nume**

* Este o colectie de tipuri sau de alte spatii de nume

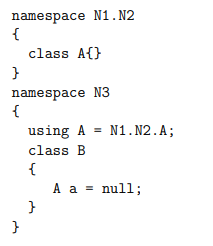
**Declaratii de spatii de nume**

* Consta in cuvantul cheie namespace
* Sunt implicit publice si acest tip de acces NU se poate modifica
* In interiorul unui spatiu de nume se pot utiliza alte spatii de nume
* Identificatorul unui spatiu de nume poate fi simplu sau o secventa de identificatori separate prin “.”
* Cea de a doua forma permite definirea de spatii de nume imbricate, fara a se imbrica efectiv

**Directiva using**

* Faciliteaza in primul rand utilizarea spatiilor de nume si a tipurilor definite in acestea
* Nu creeaza membri noi in cadrul unitatii de program
* Au rol de a usura referirea tipurilor
* Nu se pot utiliza in interiorul claselor, structurilor, enumerarilor
* Directiva using pentru import simbolic
  + Permite importarea simbolica a tuturor tipurilor continute direct intr-un spatiu de nume
  + Tipurile declarate in interiorul unui spatiu de nume pot avea modificatory de acces public sau internal, ultimul fiind modificatorul implicit. Un tip internal nu poate fi folosit prin import in afara assembly-ului, pe cand unul public, da.
* Directiva using ca alias
  + Introduce un identificator care serveste drept alias pentru un spatiu de nume sau pentru un tip



**Declararea unei clase**

* Modificatorii de clasa sunt:
  + Public
  + Internal – se poate folosi atat pentru clase imbricate, cat si pentru clase care sunt continute in spatii de nume; semnifica acces permis doar in clasa sau assembly-ul care o cuprinde
  + Protected – se poate specifica doar pentru clase imbricate,
  + Private – doar pentru clase imbricate; semnifica acces limitat la clasa continatoare; este modificatorul implicit
  + Protected internal – folosibil doar pentru clase imbricate; tipul definit este accesibil in spatiul de nume curent, in clasa continatoare sau in tipurile derivate din clasa continatoare
  + Private protected – folosibil doar pentru clase imbricate; tipul definit este accesibil in spatiul de nume curent, in clasa continatoare sau in tipurile derivate din clasa continatoare
  + New - permis pentru clasele imbricate; ascunde un membru cu acelasi nume care este mostenit
  + Sealed – o clasa sealed nu poate fi mostenita; poate fi clasa imbricate sau nu
  + Abstract – clasa care este incomplete definite si care nu poate fi instantiate; folosibila pentru clase imbricate sau continute in spatii cu acelasi nume
  + Partial – clasa este definite in mai multe fisiere

**Membrii unei clase**

* Constante
* Campuri
* Metode
* Proprietati
* Evenimente
* Indexatori
* Operatori
* Constructori(de instanta)
* Destructori
* Constructor static
* Tipuri
* Modificatori de acces:
  + Public
  + Protected
  + Internal – accesibil in assembly-ul curent
  + Protected internal – reuniunea precedentelor doua
  + Private internal – accesibil in clasa curenta sau in clase derivate din cea curenta, dar din acelasi assembly
  + Private – accesibil doar in clasa continatoare; specificator implicit

**Constructori de instanta**

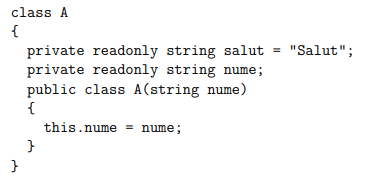
* Este un membru care implementeaza actiuni care sunt cerute pentru a initializa o instanta a unei clase
* Are acelasi nume ca si clasa din care face parte si nu returneaza un tip
* Nu se mostenesc
* Se creeaza unul implicit de catre compilator daca nu exista niciunul
* Un constructor poate apela alt constructor

**Campuri**

* Reprezinta un membru asociat cu un obiect sau cu o clasa
* E necesara precizarea unui tip de date

**Campuri readonly**

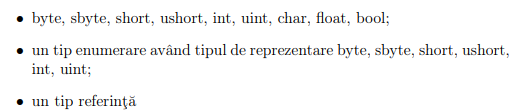
* Se face prin specificarea cuvantului readonly



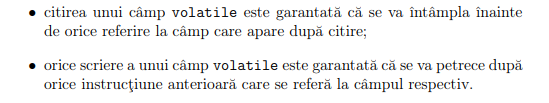
* Atribuirea se poate face doar la declararea sa sau prin intermediul unui constructor

**Campuri volatile**

* Se poate specifica doar pentru tipurile:

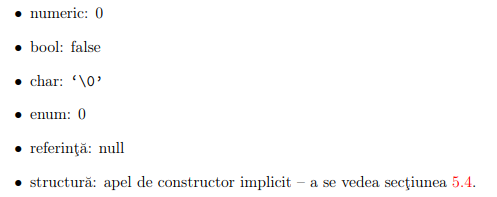


* Optimizari:



**Initializarea campurilor**

* Pentru fiecare camp se va asigna o valoare implicita astfel:



**Constante**

* Este un camp sau o entitate locala a unei metode/proprietati/indexator
* Dupa atribuire, valoarea ei nu va putea fi modificata
* Modificatori: public, protected, internal, protected internal, private
* Este automat un camp static

**Metode**

* Este un membru care implementeaza o actiune care poate fi efectuata de catre un obiect sau o clasa
* Metode externe
  + Se declara folosind modificatorul ‘extern’
  + Sunt implementate extern, de obicei in alt limbaj de programare

|  |
| --- |
| **Cap4 – Clase(continuare)** |

**Proprietati**

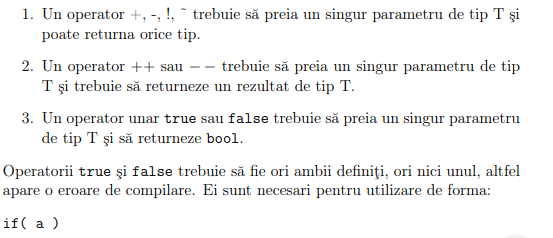
* Este un membru care permite acces la partea de stare a unei clase sau obiect
* Ex: lungimea unui sir, numele unui client, textul continut intr-un textbox
* Sunt extensii naturale ale campurilor
* Nu presupun alocarea de memorie
* Sunt metode-accesori- ce permit citirea sau setarea unor campuri ale unui obiect sau clase
* Reprezinta modalitatea de scriere a unor metode get/set pentru clase sau obiecte
* Modificatori: protected, internal, private, protected internal, private protected, public
* Un accesor get corespunde unei metode fara parametri, care returneaza o valoare
* Un accesor set corespunde unei metode cu un singur parametru de tipul proprietatii si tip de retur void; parametru mereu denumit **value**
* Tipuri:
  + Read-write – daca are ambele tipuri de accesori
  + Read-only – daca are doar accesor get
  + Write-only – daca are doar accesor set

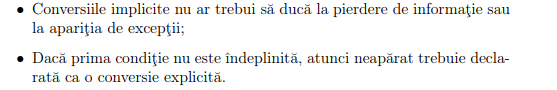
**Indexatori**

* Este o generalizare a supraincarcarii operatorului [] din c++
* Modificatori: new, public, protected, internal, private, protected internal, private protected, virtual, sealed, override, abstract, extern

**Operatori**

* Un operator este un membru care defineste semnificatia unei expresii operator care poate fi aplicata unei instante a unei clase.
* Corespunde supraincarcarii operatorilor din C++
* Reguli
  + Orice operator trebuie sa fie declarant public si static
  + Parametrii unui operator trebuie sa fie transmisi prin valoare
  + Acelasi modificator nu poate aparea de mai multe ori in antetul unui operator
* Operatori
  + Operatori unari:



* + Operatori binari
    - Operatorii sunt: + - \* / % & | ^ >> << == != > < >= <=
  + Operatori de conversie
    - 

**Constructor static**

* Este un membru care implementeaza actiunile cerute pentru initializarea unei clase
* Nu se mosteneste
* Nu se poate apela direct
* Nu se poate supraincarca
* Se va executa cel mult o data in aplicatie
* Se va apela inaintea primei creari a unei instante a clasei respective sau inaintea primului acces la un membru static
* Nu are specificator de acces
* Poate accesa doar membri static

**Clase imbricate**

* Accesarea se face prin NumeClasaExterioara.NumeClasaImbricata
* Se comporta ca un membru static al tipului continator
* Crearea unei instante nu trebuie sa fie precedata de crearea unei instante a clasei exterioare continatoare
* Nu se poate folosi this in interiorul unei clase imbricate pentru a accesa membri instanta din tipul continator
* Poate manipula toti membrii din interiorul clasei continatoare, indiferent de gradul lor de accesibilitate.
* Daca clasa continatoare are membri statici atunci ei pot fi folositi fara a se folosi numele clasei
* Se folosesc in cadrul containerilor pentru care trebuie sa se construiasca un enumerator

**Destructori**

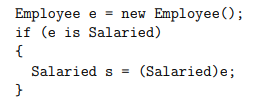
* Nu are modificator de acces
* Nu poate fi apelat manual
* Nu poate fi supraincarcat
* Nu este mostenit
* Este o scurtatura sintactica pentru metoda virtuala Finalize() definita in clasa System.Object

**Clase statice**

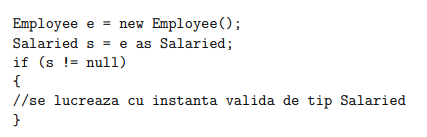
* Se foloseste atunci cand se doreste accesarea membrilor fara a fi nevoie sa se lucreze cu obiecte
* Nu pot fi instantiate
* Nu pot fi mostenite
* Contin doar membri static

**Specializarea si generalizarea**

* Specializarea:
  + - Tehnica de a obtine noi clase pornind de la cele existente
* Generalizarea
  + - Procedeul invers specializarii
* Operatorul is:
  + Este folosit pentru a verifica daca un anumit obiect este de un anumit tip
  + Este folosit de obicei inaintea operatiilor de conversie explicita de la un tip de baza la unul derivate
  + Daca obiectul nu este de tipul la care se face conversie ar rezulta o exceptie de tipul System.InvalidCastException



* Operatorul as:
  + Este folosit pentru conversii explicite
  + Returneaza un obiect de tipul la care se face conversia sau null daca conversia nu se poate face(nu se arunca exceptii)



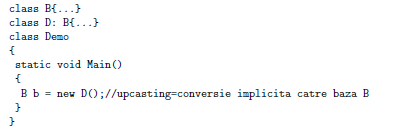
**Clase sealed**

* Specificatorul sealed se poate folosi inaintea cuvantului cheie class
* Specifica faptul ca clasa curenta nu se poate deriva
* Este eroare de compilare ca o clasa sealed sa fie declarata drept clasa de baza
* Pot duce la simplificarea modelului

|  |
| --- |
| **Cap5 – Clase – polimorfism, clase abstracte. Structuri si interfete** |

**Polimorfismul**

* Capacitatea unei entitati de a lua mai multe forme
* In C# este de 3 feluri:
  + Parametric
    - Este cea mai slaba forma de polimorfism
    - Se permite ca o implementare de metoda sa poata prelucra orice numar de parametri
    - Parametrul este trimis prin *params*
  + Ad-hoc
    - Se mai numeste si supraincarcarea metodelor
    - Mechanism prin care in cadrul unei clase se pot scrie mai multe metode avand acelasi nume, dar tipuri sau numere diferite de parametri
    - Alegerea metodei se va face la compilare
  + De mostenire
    - Forma cea mai evoluata de polimorfism
    - Este necesara existenta unei ierarhii de clase

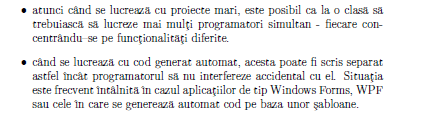
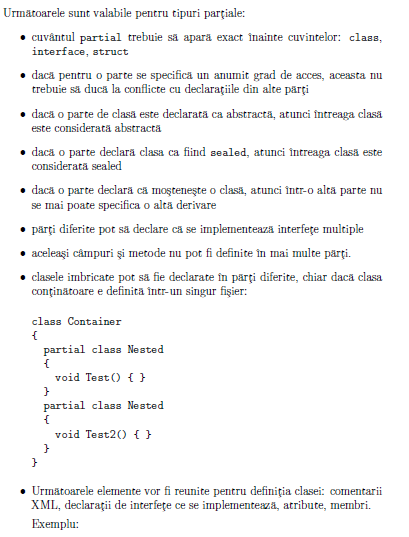


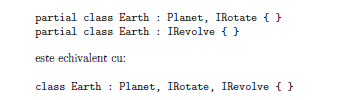
* + - Trebuie sa fie rezolvat la rulare, nu la compilare in functie de natura efectiva a obiectelor
* Virtual si override
  + Asigura executarea metodei la rulare si nu la compilare
  + In clasa se baza se declara metoda virtuala
  + In clasa derivata metoda este override
* Modificatorul new pentru metode
  + Indica faptul ca o metoda dintr-o clasa derivata care are aceeasi semnatura cu una dintr-o clasa de baza nu este o suprascriere polimorfica a ei, ci apare ca una noua
  + Daca nu se doreste suprascriere si lipseste new, se obtine avertisment de compilare(CS0108), suprascrierea ramane nepolimorfica
* Metode sealed
  + Se impiedica suprascrierea ei intr-o clasa derivata din cea curenta

**Clase si metode abstracte**

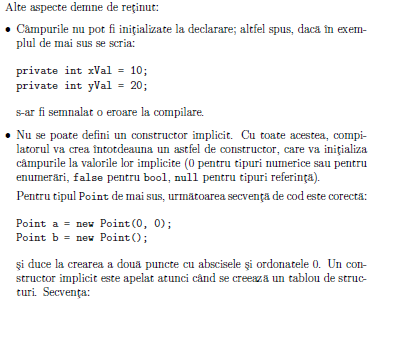
* Se foloseste cand nu are sens crearea de instante din cauza unei generalitati prea mari
* Se declara cu abstract
* Orice metoda abstracta este automat si virtuala
* Suprascrierea unei metode abstracte se face cu o metoda calificata cu override

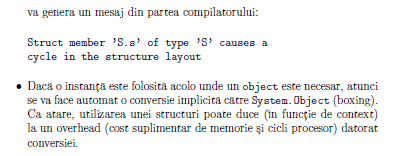
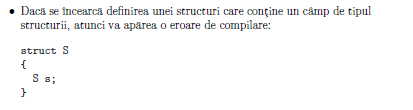
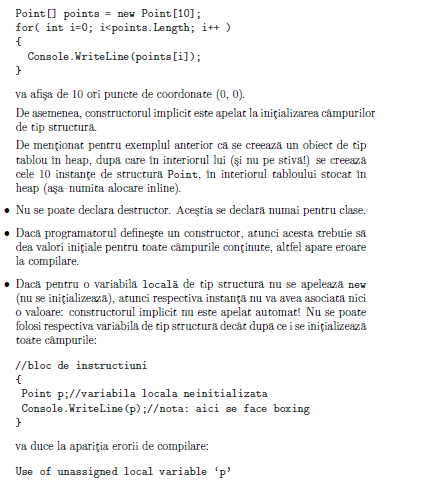
**Tipuri partiale**

* Posibilitatea de a scrie definitia unei clase, interfete sau structure in mai multe fisiere
* Beneficii:



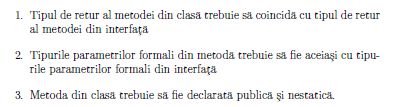
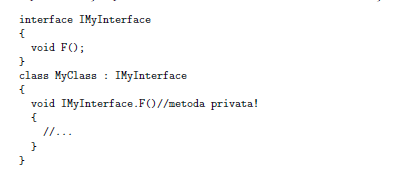
**Tipuri structura**

* Structurile reprezinta tipuri de date asemanatoare claselor
* Sunt tipuri valoare
* Versiuni mai “usoare” ale claselor
* Folosite pentru tipuri pentru care aspectul comportamental este mai putin pronuntat
* Modificatorii: new, public, protected, internal, private
* Derivata din System.ValueType -> System.Object
* Automat considerata sealed(nederivabila) -> nu se pot declara in tipurile structura membri protected, protected internal, private protected
* Poate implementa una sau mai multe interfete
* Poate contine declaratii de constante, campuri, metode, proprietati, evenimente, indexatori, operatori, constructori, constructori static, tipuri imbricate
* La atribuire se face copiere bit cu bit
* Se declara campurile ca fiind publice, sau se folosesc proprietati



**Interfete**

* Defineste un contract comportamental
* Este o relatie de implementare
* Contine: metode, proprietati, evenimente, indexatori
* Nu contine implementari pentru cele de mai sus, doar declaratii
* Modificatori: new, public, protected, internal, private
* Poate mosteni de la 0 mai multe interfete
* Corpul interfetei contine declaratii de metode, fara implementari
* Orice membru are acces public
* NU se poate specifica pentru o metoda din interior: abstract, public, protected, internal, private, virtual, override, static
* O clasa care implementeaza interfata trebuie sa implementeze toate metodele din interfata sau sa declare metodele abstracte
* Reguli:

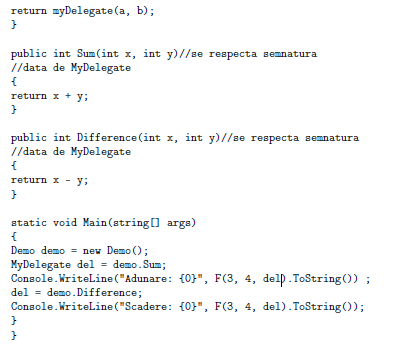
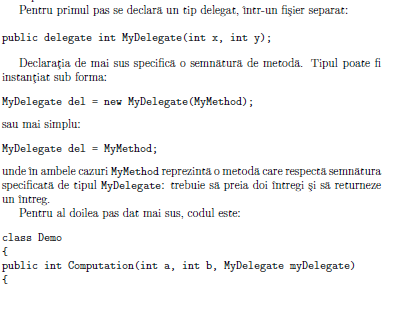


* Daca dorim sa avem metode private, folosim numele interfetei

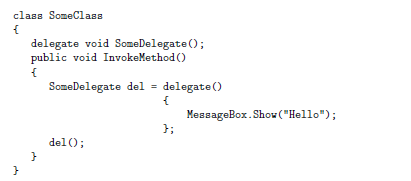
|  |
| --- |
| **Cap6 – Tipul delegat, metode anonime, evenimente, exceptii** |

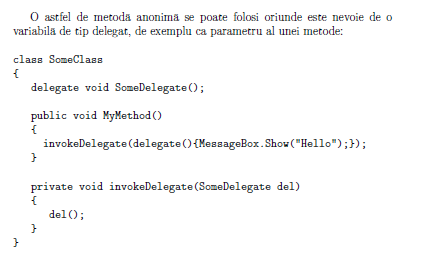
**Tipul delegat**

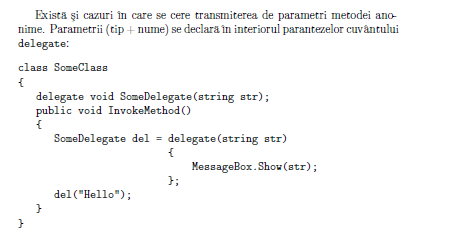
* Un delegate este un tip referinta folosit pentru a Incapsula o metoda cu un anumit antet(tipul parametrilor formali si tipul de retur).
* Modificatori: new, public, protected, internal, protected internal, private
* Derivat din System.Delegate
* Poate fi specificat atat in interiorul unei clase, cat si in exteriorul ei
* Daca e declarat in interiorul unei clase, atunci este si static



**Metode anonime**







**Multicasting**

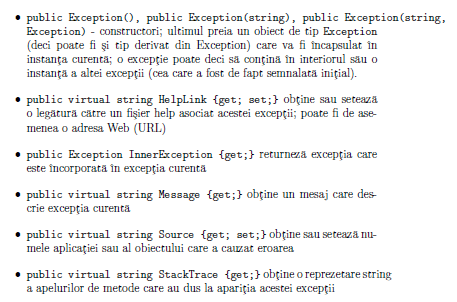
* Mecanism esential pentru tratarea evenimentelor
* Unui delegat i se poate atribui mai mult de o metoda
* Orice delegat care returneaza void este un delegat multicast
* Doi delegati multicast pot fi combinati folosind “+”
* Daca unui delegat multicast i se asociaza mai multe metode si una dintre ele arunca exceptie la rulare, atunci restul de metode din lantul de apel nu se vor mai executa

**Evenimente si delegate**

* Tratarea evenimentelor in C# se face folosind delegati multicast
* Metodele care raspund la un eveniment se numesc event handlers
* Prin conventie, un event handler in .Net returneaza void si preia doi parametri
* Modificatori de eveniment: abstract, new, public, protected, internal, private, static, virtual, sealed, override, extern

**Tratarea exceptiilor**

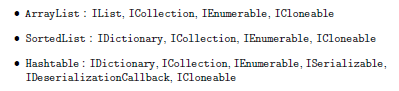
* Tipul exception
  + Se arunca obiecte de tip System.Exception

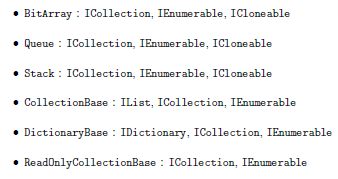


* Aruncarea si prinderea exceptiilor
  + Aruncarea cu throw
  + Aruncarea unei exceptii opreste executia metodei curente

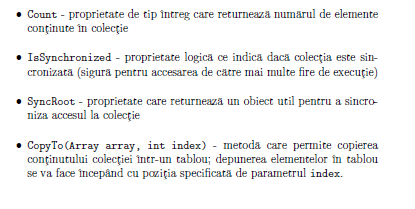
|  |
| --- |
| **Cap7 – Colectii si tipuri generice** |

**Colectii**

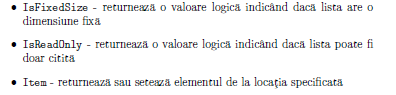
* Principalele colectii impreuna cu interfetele implementate sunt:



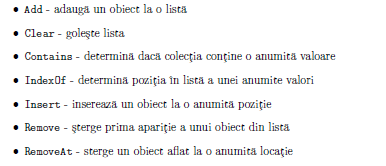
* IEnumerable
  + Permit iterarea peste o colectie de elemente
  + Metoda: GetEnumerator()
* ICollection
  + Tipul de baza pentru orice clasa de tip colectie
  + Proprietati si metode:



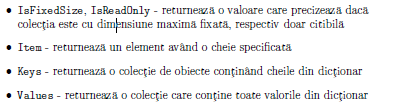
* IList
  + Colectie de obiecte care pot fi accesate individual printr-un index intreg
  + Proprietati:



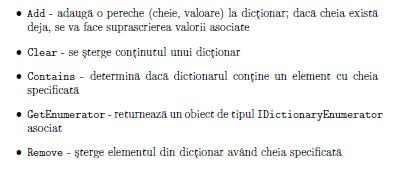
* + Metode:



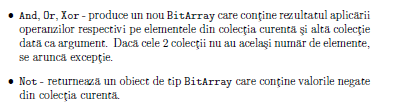
* IDictionary
  + O colectie de perechi(cheie, valoare)
  + Permite indexarea unei colectii de elemente dupa altceva decat indici intregi
  + Fiecare pereche trebuie sa aiba o cheie unica
  + Proprietati:



* + Metode



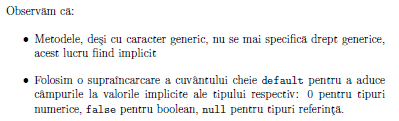
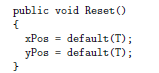
* Iteratori pentru colectii
* Colectii de tip lista
  + ArrayList
    - Este o clasa concreta(instantiabila)
    - Stocheaza o colectie de elemente sub forma unui vector auto-redimensionabil
    - Suporta mai multi cititori concurenti
    - Poate fi accesat ca un vector, folosind un indice de pozitie
  + BitArray
    - Gestioneaza un vector de elemente binare reprezentate ca booleeni, unde true = 1, false = 0
    - Metode:



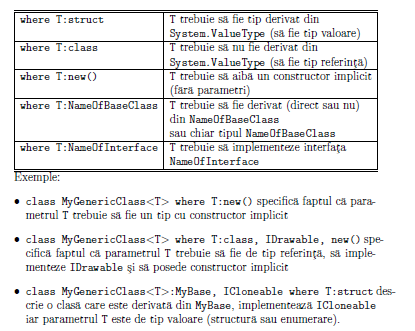
* + Stack
    - Reprezinta o colectie ce permite lucrul conform principiului LIFO
  + Queue
    - Implementeaza politica FIFO
  + CollectionBase
    - Reprezinta o clasa abstracta, baza pentru o colectie puternic tipizata
* Colectii de tip dictionar: contin obiecte care manipuleaza prin intermediul cheii associate, extind interfata IDictionary
  + SortedList
    - Reprezinta o colectie de perechi de tip (cheie, valoare) care sunt sortate dupa cheie
    - Se pot accesa dupa cheie sau dupa index
  + Hashtable
    - Reprezinta o colectie de perechi de tip (cheie, valoare) care e organizata pe baza codului de dispersie(hashing) al cheii.
    - O cheie nu poate fi nula
    - Obiectele folosite pe post de chei trebuie sa suprascrie metodele Object.GetHashCode si Object.Equals
    - Obiectele folosite pe post de cheie trebuie sa fie imuabile
  + DictionaryBase
    - Reprezinta o clasa de baza abstracta pentru implementarea unui dictionar utilizator puternic tipizat

**Clase generice**

* Sunt blocuri de cod parametrizate care permit scrierea unui cod general, ce poate fi ulterior adaptat automat la cerintele specific ale programatorului
* Metode generice
  + Tipul generic T poate fi folosit si ca tip de retur al metodei generice, dar cel putin un parametru formal trebuie sa fie de tip T
* Tipuri generice



* Restrictii:



* Interfete si delegati generic





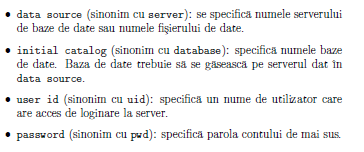
|  |
| --- |
| **Cap8 – ADO.NET** |

**Componentele unui furnizor de date**

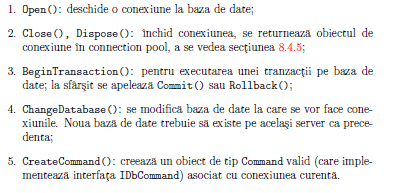
* Connection
  + Folosite pentru a reprezenta o conexiune la sursa de date
  + Contin locatia sursei, numele si parola
* Command
  + Folosite pentru a executa diferite comenzi pe baza de date(Select, Insert, Update, Delete)
  + Furnizeaza un obiect de tip DataReader
  + Folosite pentru apelarea procedurilor stocate
  + Permit scrierea de interogari SQL parametrizate
* DataReader
  + Permit navigarea de tip forward-only, read-only si in mod conectat la sursa de date
  + Se obtin pe baza unui obiect de tip Command prin apelul metodei ExecuteReader()
* DataAdapter
  + Functioneaza ca punte intre sursa de date si obiecte de tip DataSet deconectate
  + Permit prelucrarea deconectata a datelor si ulterior reflectarea modificarilor pe baza de date
  + Contin referinte catre obiecte de tip Connection
  + Pot deschide/inchide singure conexiunea la baza de date
  + Contine referinte catre patru comenzi pentru selectare, stergere, modificare si adaugare in baza de date
* DataSet
  + Nu este parte a unui furnizor de date .NET, e independenta
  + Are avantaj ca poate lucra deconectat de la sursa de date
  + Un obiect DataSet este un container de tabele si relatii intre tabele
  + Un obiect DataSet foloseste serviciile unui obiect de tip DataAdapter pentru a-si procura datele si a trimite modificarile inapoi catre baza de date

**Obiecte Connection**

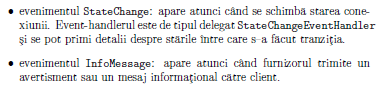
* Proprietati:
  + ConnectionString: de tip String, cu accesori get si set
    - Refera un sir de caractere ce contine detalii despre locatia sursei de date la care se face conectarea si eventual contul si parola de acces
    - String-ul de conexiune contine lista de parametri necesari pentru conectare
    - Parametri:



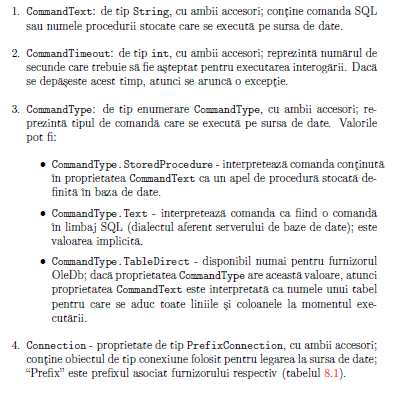
* + - Valoarea unui string de conexiune poate fi setata doar daca conexiunea e inchisa
  + ConnectionTimeout: de tip int, cu accesor get, valoare implicita 15
    - Specifica numarul de secunde pentru care un obiect de conexiune ar trebui sa astepte pentru realizarea conectarii la server inainte de a genera o exceptie
  + Database: de tip string, read-only
    - Returneaza numele bazei de date la care s-a facut conectarea
    - Folosita pentru a arata unui utilizator care este baza de date pe care se face operarea
  + Provider: de tip string, read-only
    - Returneaza numele furnizorului OLE DB
  + ServerVersion: de tip string, read-only
    - Returneaza versiunea de server la care s-a facut conectarea
  + State: de tip enumerare ConnectionState, read-only
    - Returneaza starea curenta a conexiunii
    - Valori posibile:
      * Broken
      * Closed
      * Connecting
      * Executing
      * Fetching
      * Open
* Metode

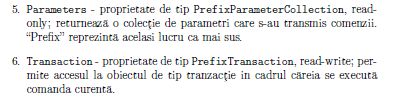


* Evenimente



**Obiecte Command**

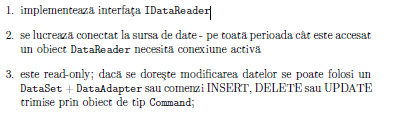
* O clasa de tip Command trebuie sa implementeze interfata IDbCommand
* Este folosit pentru a executa comenzi pe baza de date: SELECT, INSERT, DELETE, UPDATE sau apel de proceduri stocate
* Comanda se poate executa numai daca s-a deschis o conexiune la baza de date
* Proprietati:



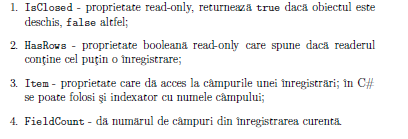
* Metode:
  + Constructori: un obiect de tip comanda poate fi creat si prin intermediul apelului de constructor
  + Cancel()
    - Incearca sa opreasca o comanda daca ea se afla in executie
    - Daca nu se afla in executie atunci nu se intampla nimic
  + Dispose()
    - Disponibilizeaza obiectul comanda
  + ExecuteNonQuery()
    - Executa o comanda care nu returneaza un set de date din baza de date
    - Daca comanda a fost de tip INSERT, UPDATE, DELETE, se returneaza numarul de inregistrari afectate
    - Daca nu e definita conexiunea la baza de date sau aceasta nu e deschisa, se arunca exceptie de tipul InvalidOperationException
  + ExecuteReader()
    - Executa comanda continuta in proprietatea CommandText
    - Se returneaza un obiect de tip IDataReader
    - Poate lua un argument optional de tip enumerare CommandBehavior:
      * CommandBehavior.CloseConnection – conexiunea este inchisa atunci cand obiectul de tip IDataReader este inchis
      * CommandBehavior.KeyInfo – comanda returneaza metadate despre coloane si cheia primara
      * CommandBehavior.SchemaOnly – comanda returneaza doar informatii despre coloane
      * CommandBehavior.SequentialAccess – da posibilitatea unui obiect DataReader sa manipuleze inregistrari care contin campuri cu valori binare de mare intindere
      * CommandBehavior.SingleResult – se returneaza un singur set de rezultate
      * CommandBehavior.SingleRow – se returneaza o singura linie
  + ExecuteScalar()
    - Executa comanda continuta in proprietatea CommandText
    - Se returneaza valoarea primei coloane de pe primul rand
    - Folosit pentru obtinerea unor rezultate de tip agregat
  + ExecuteXmlReader()
    - Returneaza un obiect de tipul XmlReader obtinut din rezultatul interogarii pe sursa de date

**Obiecte DataReader**

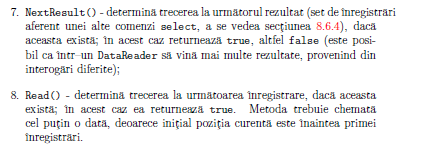
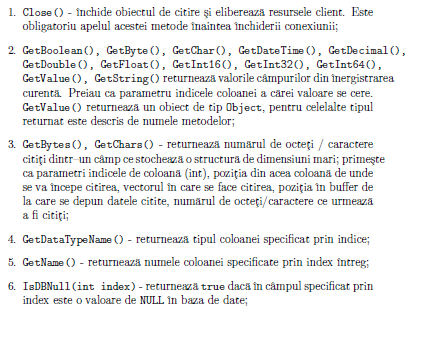
* Este folosit pentru a citi date dintr-o sursa de date
* Caracteristici:



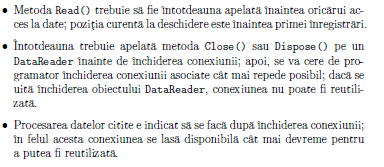
* Avantaje:
  + Accesul conectat
  + Performantele bune
  + Consumul mic de resurse
  + Tipizarea puternica
* Proprietati:



* Metode



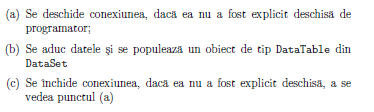
* Crearea si utilizarea unui obiect DataReader



|  |
| --- |
| **Cap9 – ADO.NET(II)** |

**Obiecte DataAdapter**

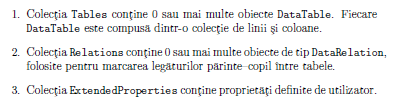
* Orice clasa de tipul DataAdapter este derivate din clasa abstracta DbDataAdapter
* Metode:
  + Constructori: de la cel implicit(fara parametri) pana la cei in care se specifica o comanda de tip SELECT si conexiunea la sursa de date
  + Fill()
    - Metoda polimorfica
    - Permite umplerea unei tabele dintr-un obiect de tip DataSet cu date
    - Permite specificarea obiectului DataSet in care se depun datele, numarul de inregistrare cu care sa se inceapa popularea, numarul de inregistrari care urmeaza a fi aduse
    - La apelare:



* + Update()
    - Metoda polimorfica
    - Permite reflectarea modificarilor effectuate intr-un DataSet
    - Se returneaza numarul de inregistrari afectate
* Proprietati
  + DeleteCommand, InsertCommand, SelectCommand, UpdateCommand – de tip Command
    - Contin comenzile ce se executa pentru selectarea sau modificarea datelor in sursa de date
  + MissingSchemaAction – de tip enumerare MissingSchemaAction
    - Determina ce se face atunci cand datele care sunt aduse nu se potrivesc peste schema tabelei din obiectul DataSet
    - Poate avea valorile:
      * MissingSchemaAction.Add – implicit, DataAdapter, adauga coloanele la schema tabelei
      * MissingSchemaAction.AddWithKey – ca mai sus, dau nu adauga si metadate relative la cine este cheia primara
      * MissingSchemaAction.Ignore – se ignora nepotrivirea dintre coloanele aduse din baza de date si cele existente in obiectul DataSet
      * MissingSchemaAction.Error – se genereaza o exceptie de tipul InvalidOperationException

**Clasa Dataset**

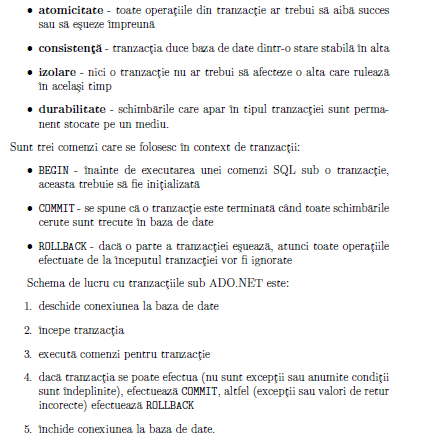
* Nu face parte din biblioteca unui furnizor de date ADO.NET
* Face parte din .NET framework
* Poate sa contina reprezentari tabelare ale datelor din baza precum si diferite restrictii si relatii existente intre tabele
* Permite lucrul deconectat de la sursa de date, eliminand necesitatea unei conexiuni permanent deschise la baza de date precum la DataReader
* Continut:



* Clasa DataTable
  + Datele sunt continute intr-un dataset sub forma unor tabele de tip DataTable
  + Pot fi folosite atat independent, cat si in interiorul unui DataSet ca elemente ale colectiei Tables
  + DataColumn
    - Defineste numele si tipul unei coloane care face parte din sau se adauga unui obiect DataTable
  + DataRow
    - Reprezinta o linie dintr-un obiect DataTable
    - Pentru crearea unei linii se poate apela metoda NewRow() pentru o tabela a carei schema se cunoaste
  + Constrangeri
    - Sunt folosite pentru a descrie anumite restrictii aplicate asupra valorilor din coloanele unei tabele
    - UniqueConstraint
      * Precizeaza ca intr-o anumita coloanal valorile sunt unice
    - ForeignKeyConstraint
      * Specifica actiunea care se va efectua atunci cand se sterge o inregistrare sau se modifica valoarea dintr-o anumita coloanal
  + Stabilirea cheii primare
    - O cheie primara se defineste ca un tablou unidimensional de coloane care se atribuie proprietatii PrimaryKey a unei tabele

**Tranzactii in ADO.NET**

* O tranzactie este un set de operatii care se efectueaza fie in intregime, fie deloc
* Proprietatile ACID

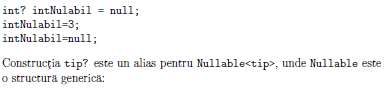


**Lucru generic cu furnizori de date**

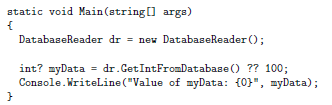
* Clase de baza pentru tipurile folosite intr-un furnizor de date:
  + DbCommand – clasa abstracta, baza pentru clasele Command
  + DbConnection – clasa abstracta, baza pentru clasele Connection
  + DbDataAdapter – clasa abstracta, baza pentru clasele DataAdapter
  + DbDataReader – clasa abstracta, baza pentru clasele DataReader
  + DbParameter – clasa abstracta, baza pentru clasele Parameter
  + DbTransaction – clasa abstracta, baza pentru clasele Transaction

**Tipuri nulabile**

* Mecanism de extindere a tipurilor valoare astfel incat sa suporte si valoarea nul
* Exemplu pentru o variabila de tip int care sa poata avea si valoare nula:



* In contextual tipurilor ce suporta valori de nul s-a introdus operatorul ?? care permite asignarea unei valori pentru o variabila de tip nulabil daca valoarea returnata este nula

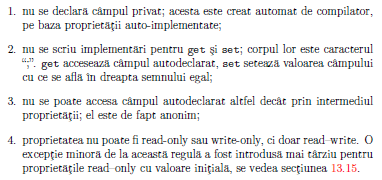


* HasValue – verifica daca o variabila de tip nulabil are valoare sau contine null

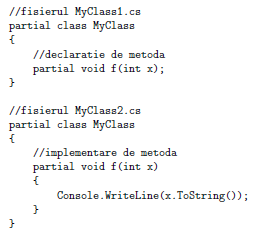
|  |
| --- |
| **Cap10 - LINQ** |

**Elemente specific C#**

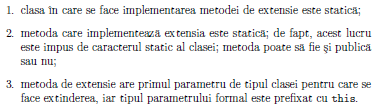
* Proprietati implementate automat
  + Particularitati:



* + - Daca ulterior se decide implementarea unui accesor de catre programator, atunci si celalalt trebuie implementat, iar campul privat trebuie declarat in mod explicit
  + Initializatori de obiecte
    - Membrii pentru care se face initializare trebuie sa fie publici
    - In particular, ei pot fi si campuri, dar acest lucru nu este incurajat de principiul incapsularii
  + Inferenta tipului
    - Var arata ca la compilare se poate deduce tipul actual al variabilelor locale respective.
    - Var este o scurtatura
    - Conditii pentru folosirea var:
      * Se poate folosi doar pentru variabilele locale
      * Se poate folosi doar atunci cand compilatorul poate sa infereze tipul de data asociat variabilei
      * Odata ce compilatorul determina tipul de data, acesta nu mai poate fi schimbat
  + Tipuri anonime
    - Permite declararea unor variabile de un tip care nu este definit aprioric
    - Se omite declararea numelui de clasa si a componentei acesteia
    - Proprietatile sunt publice si read-only
  + Metode partiale
    - Sunt metodele care sunt declarate in mai multe parti ale unei clase
    - Clasa continatoare poate sau nu sa fie partiala
    - Reguli:
      * Metoda pentru care se foloseste implementare partiala trebuie sa returneze void
      * Parametrii nu pot fi transmisi prin out
      * Metoda nu poate avea specificatori de acces, ea este private
      * Metoda trebuie sa fie declarata atat la definire cat si la implementare ca fiind partiala



* + - Sunt utile atunci cand anumiti pasi dintr-un flux de lucru sunt optionali(validari, transformari de date)
  + Metode de extensie
    - Permit scrierea de metode associate cu clase, alte clase decat cele in care sunt definite
    - Se poate utiliza daca:

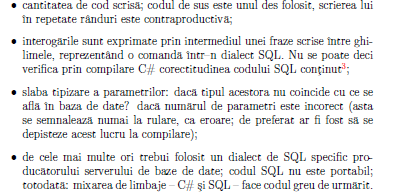


**Generalitati**

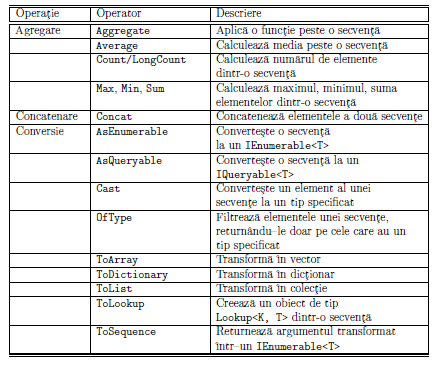
* LINQ permite interogarea unor colectii de date folosind o sintaxa integrata in plarforma .NET
* Se pot interoga colectii de forma: tablouri, colectii, clase enumerabile, documente XML, baze de date relationale
* Datele rezultate sunt vazute ca niste obiecte
* Are loc o asociere a unor date neobiectuale intr-un format usor de folosit in limbajele obiectuale din cadrul platformei
* Interfata IQueryable<T>
  + Permite implementarea unor furnizori LINQ
  + Expresia folosita pentru interogare este tradusa intr-un arbore de expresie
* LINQ este alcatuit din:
  + LINQ to Objects
  + LINQ to XML
  + LINQ to DataSets
  + LINQ to Entities

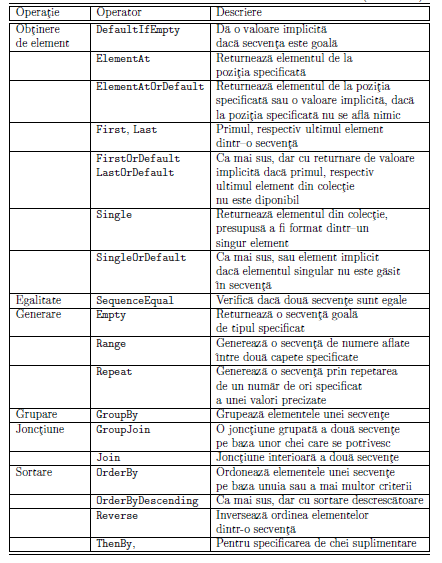
**Motivatie**

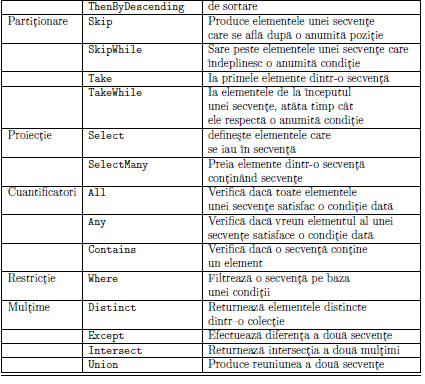
* Motive pentru care LINQ este util:
  + Codul stufos, neproductiv utilizat pentru accesarea in modul clasic al datelor
  + Nepotrivirea paradigmelor obiectual-relationale
* Codul clasic ADO.NET



Operatori LINQ







|  |
| --- |
| **Cap11 – LINQ(II)** |

**LINQ to Objects**

* Operatorul let
  + Permite evaluarea unei expresii si atribuirea rezultatului catre o variabila care poate fi utilizata in cadrul interogarii ce o contine
* Operatii de ordonare
  + OrderBy
    - Produce sortarea crescatoare



* + OrderByDescending
    - Produce sortarea descrescatoare



* + ThenBy
    - Specificarea mai multor chei dupa care sa se faca sortarea





* + ThenByDescending
    - Specificarea mai multor chei dupa care sa se faca sortarea
  + Reverse
    - Inversarea ordinii elementelor dintr-o secventa

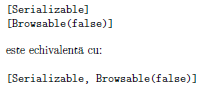


* Paginare
  + Take
  + Skip
  + TakeWhile
  + SkipWhile
* Concatenarea
  + Se face cu concat
  + Preia doua secvente si o produce pe a treia
  + Nu elimina duplicatele
* Referirea de elemente din secvente
  + First
  + FirstOrDefault
  + Last
  + LastOrDefault
  + Single
  + SingleOrDefault
  + ElementAt
  + ElementAtOrDefault
  + DefautIfEmpty
* Jonctiuni
  + Operatorul Join poate fi folosit pentru a adduce date din doua colectii , date care sunt puse in corespondenta pe baza unei egalitati de valori
  + Sunt aduse foar acele elemente din fiecare colectie care au un correspondent in cealalta colectie
* Grupare
  + Se face cu GroupBy
  + Se grupeaza elementele unei secvente pe baza unui selector
* Agregare
  + Count
    - Returneaza un intreg
  + LongCount
    - Returneaza un long
  + Sum
  + Min
  + Max
  + Average
  + Aggregate
  + Operatorii Min, Max, Average trebuie apelati doar pentru o colectie de valori numerice

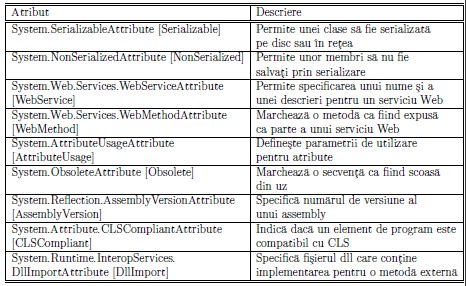
|  |
| --- |
| **Cap12 – Atribute si fire de executie** |

**Atribute**

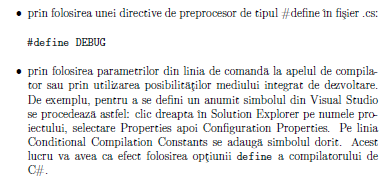
* Sunt folosite pentru a da o extra-informatie compilatorului de .NET
* Utile in urmarirea bug-urilor care exista intr-un sistem sau urmarirea stadiului proiectului
* Sunt de 2 feluri
  + Intrinseci(predefinite)
    - Integrate in platforma .NET
  + Definite de programator
* Se pot specifica pentru:
  + Assembly-uri
  + Clase
  + Constructori
  + Delegate
  + Enumerari
  + Evenimente
  + Campuri
  + Interfete
  + Metode
  + Module
  + Parametri
  + Proprietati
  + Indexatori
  + Valori de retur
  + Tipuri structura
* Se specifica prin paranteze drepte
* Mai multe attribute se scriu unul sub altul sau despartite prin virgula in aceeasi paranteza



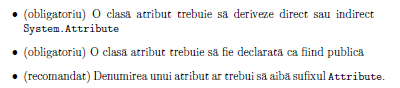
* Atribute predefinite



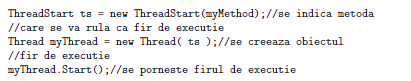
* Alte attribute
  + Conditional
    - Se ataseaza la o metoda pentru care se doreste ca atunci cand compilatorul o intalneste la un apel, daca un anumit simbol nu e definit, atunci nu va fi chemata
    - Este folosita pentru a omite anumite apeluri de metode(de ex cele folosite la etapa de debugging)
    - Definirea unui simbol se face in 2 feluri



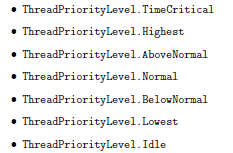
* + CLSCompliant
    - Se aplica pe assembly-uri
* Atribute definite de programator
  + E o clasa derivata din System.Attribute
  + Pasi:
    - 1. Declararea tintei
    - 2. Declararea unei clase atribut
      * Reguli:



* + - 3. Declararea constructorilor si a proprietatilor
    - 4. Utilizarea atributului
* Fire de executie
  + Sunt responsabile cu multitasking-ul in interiorul unei aplicatii
  + Se gasesc in spatiul de nume System.Threading
* Managementul thread-urilor
  + Pornirea thread-urilor
    - Cea mai simpla metoda de a crea un fir de executie este de a crea o instanta a clasei Thread, al carei constructor preia un singur argument de tip delegate
    - Crearea unui fir de executie se face pe baza unei metode care returneaza void si nu preia niciun parametru



* + - Timpul fiecarui thread este dat de planificatorul de thread-uri si depinde de procesor
    - Thread.CurrentThread -> referinta catre thread-ul curent
  + Metoda Join()
    - Determina suspendarea thread-ului curent si pornirea celui care a apelat join
  + Suspendarea firelor de executie
    - Se face cu metoda statica Sleep()
    - Sleep() poate prelua un parametru de tip intreg care reprezinta milisecundele de “adormire”
    - Sleep() poate prelua un argument de tip TimeSpan(cea mai mica unitate de timp care poate fi specificata, egala cu 100 de nanosecunde)
  + Oprirea thread-urilor
    - Se face cu metoda Abort()
    - Se transmite o exceptie in firul de executie: ThreadAbortException
  + Sugerarea prioritatilor firelor de executie
    - Se seteaza cu ThreadPriorityLevel -> enum
    - Implicit e Normal
    - Niveluri:



* + - Prioritatea unui thread se seteaza cu proprietatea Priority



* + Fire in fundal si fire in prim plan
    - Un fir de executie poate sa se execute in fundal si in prim plan
    - Atata timp cat exista fire de executie in foreground, CLR mentine aplicatia in executie
    - Daca nu mai exista fire in foreground, atunci CLR termina procesul, stergand firele de executie din background daca mai exista, apeland metoda Abort()

**Sincronizarea**

* Se ocupa cu controlarea accesului la resurse partajate de mai multe fire de executie
* Mecanisme:
  + Clasa Interlock
  + Instructiunea C# lock
  + Clasa Monitor
* Clasa Interlocked
  + Include doua metode statice
    - Increment()
    - Decrement()
* Instructiunea lock
  + Marcheaza o sectiune critica a codului, producand astfel sincronizare pentru obiect
  + Este inlaturat la sfarsitul instructiunii
* Clasa Monitor
  + Contine metode pentru a controla sincronizarea firelor de executie, permitand declararea unei zone critice in care la un moment dat doar un thread trebuie sa opereze
  + Cand dorim sa se inceapa sincronizarea, se apeleaza metoda Enter
  + Metoda Wait()
    - elibereaza monitorul
    - blocheaza thread-ul
    - informeaza CLR ca atunci cand monitorul devine din nou liber, thread-ul curent ar vrea sa isi continue executia(este adaugat intr-o coada de asteptare)
  + Metoda Exit() – terminarea zonei critice
  + Metoda Pulse() – semnaleaza ca a avut loc o schimbare de stare
  + Metoda PulseAll() – anunta toate obiectele blocate de schimbarea de stare

|  |
| --- |
| **Cap13 – Noutati in .NET Framework 4.x** |

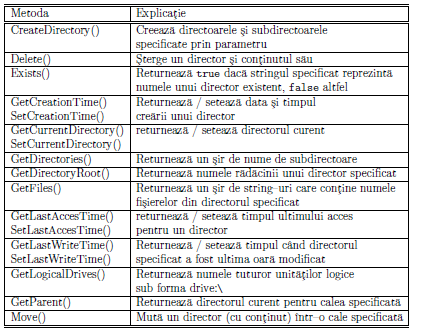
**Clasa Tuple**

* Un tuplu poate avea maxim 8 obiecte
* Se apeleaza prin apelarea Create

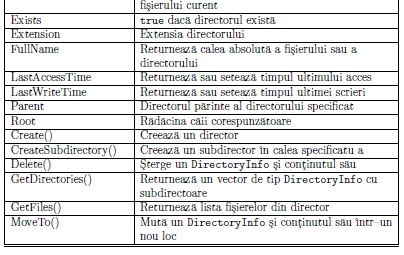
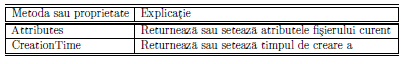
|  |
| --- |
| **Cap14 - Fluxuri** |

**Sistemul de fisiere**

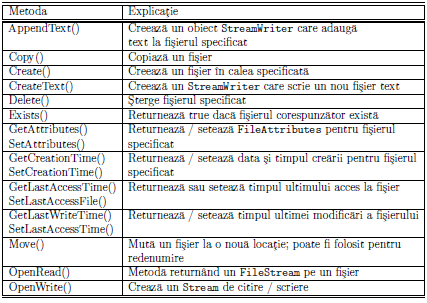
* Clasele pentru manipularea fisierelor si a directoarelor se afla in System.IO
* Clase:
  + Directory
    - Contine metode statice pentru crearea, mutarea, explorarea directoarelor
    - Metode:



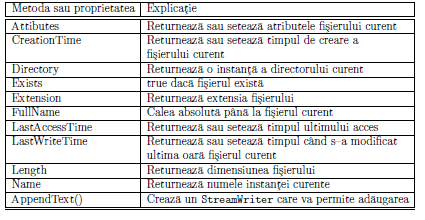
* + DirectoryInfo
    - Contine membri nestatici
    - Permite aflarea diferitelor informatii pentru un director anume
    - Metode:

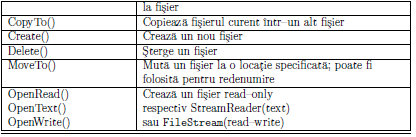


* + File
    - Are doar membri static
    - Metode:

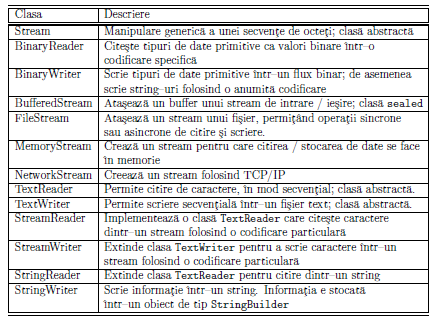


* + FileInfo
    - Are doar membri nestatici
    - Metode:

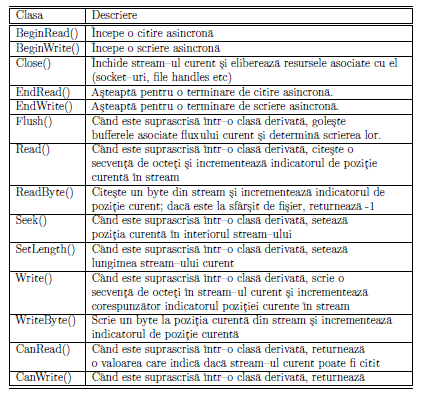


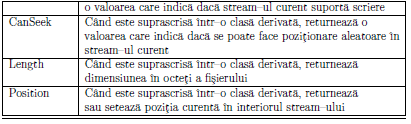


**Citirea si scrierea datelor**

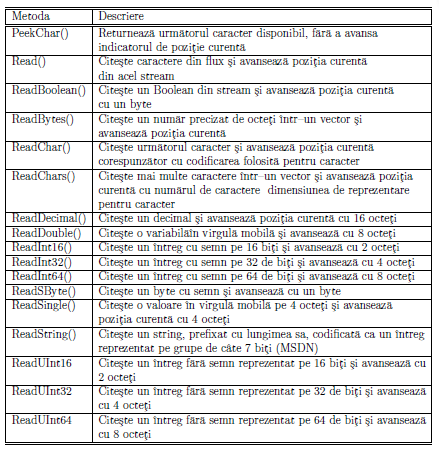


* Clasa Stream
  + Este o clasa abstracta din care se deriveaza toate celelalte clase
  + Metodele permit citire de octeti, inchidere, golire de buffer
  + Permite operatii sincrone si asincrone
  + Metode si proprietati:





* Clasa FileStream
  + Are 9 constructori supraincarcati
* Clasa MemoryStream
  + Are 7 constructori
  + Opereaza cu datele din memorie, ca si cum le-ar citi sau scrie dintr-un/intr-un fisier de pe disc.
* Clasa BufferedStream
  + Asigura o zona tampon in cazul operatiilor de intrare-iesire
  + Constructorul primeste o instanta a clasei Stream
  + Metoda Flush() – determina golirea bufferului
* Clasele BinaryReader si BinaryWriter
  + Sunt folosite pentru a accesa date mai complexe decat un byte
  + Metode BinaryReader



* Clasele TextReader, TextWriter si descendentele lor
  + TextReader
    - StreamReader – folosit pentru a scrie si citi siruri de caractere
    - StringReader – permite atasarea unor fluxuri la siruri de caractere
  + TextWriter
    - StreamWriter – folosit pentru a scrie si citi siruri de caractere
    - StringWriter – permite atasarea unor fluxuri la siruri de caractere
    - System.Web.HttpWriter
    - System.Web.UI.HtmlTextWriter – scrie o secventa HTML pe o pagina Web, folosit in script-uri
    - System.CodeDom.Compiler.IndentedTextWriter – defineste metode care insereaza tab-uri si pastreaza evidenta nivelului de identare
* Date tranziente – date care nu se salveaza, se marcheaza cu [NonSerialized]