### Modulul 10. Structuri, enumerări și tipuri nullable.

#### Structurile

1. Ce este o structură?

Structurile sunt un tip de date care pot stoca variabile, metode, proprietăți, indexatori și evenimente.

Structurile se utilizează, cel mai des, pentru a stoca date pentru că acestea sunt mai performante și ocupă mai puțină memorie decât clasele. Este posibil să folosim o structură cum folosim o clasă, adică să includem metode și logică mai complexă în interiorul lor dar acest lucru nu este recomandat pentru că structurile au o funcționalitate mai redusă, mai ales faptul că structurile nu pot moșteni de la alte clase și structuri și viceversa.

|  |
| --- |
| struct Librarie  {  public string[] titlu;  public string[] autor;  public string[] subiect;  public int[] id\_carte;  }; |

*Structură folosită pentru stocarea informației despre cărțile dintr-o librărie*

1. Sintaxa declarării unei structuri

Pentru a declara o structură vom folosi cuvântul cheie **struct**, exact cum folosi cuvântul cheie **class** când declarăm o clasă.

**Sintaxă:**

|  |
| --- |
| **[nivel de accesibilitate] struct [denumire]**  {  int numar;  }; |

**Observație:** Dacă nu veți indica un nivel de acces pentru o structură ea va fi setată automat ca publică.

1. Structuri vs clase

**Diferențe:**

1. Diferența principală între structuri și clase este că acestea fac parte din două tipuri diferite de date. **O structură este de tip valoare,** în timp ce clasele fac parte din tipul referință.

**Explicație:**

|  |
| --- |
| **Variabilele de tip valoare** au un spațiu în memorie prealocat în stivă, ceea ce înseamnă că odată ce variabila și tipul acesteia a fost definită nu putem modifica tipul ei și nu putem folosi un destructor pentru a elibera această regiune de memorie, ceea ce este caracteristic pentru structuri.  **Variabilele de tip referință** sunt flexibile și dinamice, ceea ce înseamnă că o variabilă de acest tip își poate modifica tipul de date, poate fi ințializată și distrusă, iar memoria alocată acesteia se va modifica pe parcursul funcționării programului ceea ce oferă avantajul legat de flexibilitate dar necesită un timp mai mare de procesare. |

*Diferența între tipul de date valoare și referință*

1. Structurile nu pot moșteni (doar de la interfețe) și nu pot fi moștenite.
2. Structurile nu pot folosi **constructor fără parametri**.
3. Structurile nu pot folosi destructori.
4. Variabilele structurii nu pot fi abstracte, virtuale sau protected.
5. Structurile nu pot fi inițializate fără cuvântul cheie new.
6. O structură fără date nu ocupă memorie, o clasă fără date ocupă 1 byte.
7. Necesitatea utilizării structurilor și exemple

În urma argumentelor aduse conchidem că structurile vor fi folosite pentru stocarea datelor atunci când nu avem nevoie să realocăm memorie, nu avem nevoie să moștenim de la ele și avem nevoie de un nivel mai înalt de performanță decât cel care îl oferă clasele.

Din acest punct de vedere structurile pot fi văzute ca niște blocuri rigide de date.

1. Constructorii structurilor

Ca și clasele, structurile în C# pot conține constructori. Dar, spre deosebire de clase constructorii structurilor trebuie să conțină parametri. Motivul acestui lucru este că atunci când creăm o structură, aceasta trebuie deja să conțină toate variabilele inițializate pentru a pre-aloca memorie pentru acestea în stivă.

**Exemplu:**

|  |
| --- |
| using System;  using static System.Console;  namespace StructExemplu  {  class Exemplu  {  private static void Main()  {  Angajat ang1 = new Angajat(5123, "Vasile", "Vasilievici", 3500);  WriteLine("Angajatul {0}, {1} {2} are salariul {3} lei.", ang1.id, ang1.nume, ang1.prenume, ang1.salariu);  }  }  struct Angajat  {  public int id;  public string nume;  public string prenume;  public int salariu;  public Angajat(int \_id, string \_nume, string \_prenume, int \_salariu)  {  id = \_id;  nume = \_nume;  prenume = \_prenume;  salariu = \_salariu;  }  }  } |

În exemplul de mai sus am prezentat cum poate fi folosită o structură Angajat pentru a stoca informațiile despre un angajat individual

Probleme pentru rezolvare.

### **Program 1:**

### Folosiți exemplul de mai sus dar modificați ca structura să conțină informația despre 10 angajați în tablouri unidimensionale, id[10], nume[10] etc.

1. În clasa de bază folosiți un ciclu for în care veți folosi constructorul și veți afișa informațiile despre angajați.

**Opțional**:

1. În clasa de execuție, creați câteva masive unidimensionale care vor conține, nume, prenume, id și salarii aleatorii. Utilizați aceste valori aleatorii pentru a genera obiectele de tip Angajat.

|  |
| --- |
|  |

### **Program 2:**

1. Creați un program în care veți avea o structură Numere ca va conține un tablou unidimensional de 30 numere. Creați un constructor cu parametri în interiorul structurii.
2. În clasa execuți folosiți un ciclu în care veți atribui valorile tabloului de numere cu logica următoare:
3. Numerele 1-10. Aleatorii (10-50), afișate la consolă cu culoare galbenă.
4. Numerele 11-20. Aleatorii (100-150), afișate la consolă cu culoare roz.
5. Numerele 21-30. Aleatorii(1-100), afișate la consolă cu verde.

Afișați la consolă conținutul final al tabloului.

1. Pentru fiecare interval(1-10, 11-20, 21-30) identificați și afișați maximul cu ajutorul funcțiilor LINQ.
2. Identificați maximul pentru întregul tablou cu ajutorul funcțiilor LINQ.

#### Enumerările (enum)

1. Ce este o enumerare?

Enumerările în C# reprezintă un tablou de constante de tip integer care au un anumit nume atribuit acestora.

**Exemplu:**

|  |
| --- |
| namespace EnumExemplu  {  class Exemplu  {  enum Zile  {  Luni,  Marti,  Miercuri,  Joi,  Vineri,  Simbata,  Duminica  }  private static void Main()  {  WriteLine(Zile.Luni);  WriteLine((int)Zile.Luni);  }  }  } |

În exemplul de mai sus am declarat enumerarea zile în care am setat toate zilele săptămânii.

În continuare în metoda main am afișat la consolă valoarea string a zilei de luni care este Luni și valoarea int a zilei de luni care este 0.

Enumerările se folosesc atunci când vrem să prestabilim unele stări sau valori care vor rămâne neschimbate, de exemplu starea programului (execuție, sleep, etc) sau zilele săptămânii, sau statutul unei tranzacții(în procesare, executată, anulată, etc).

Din motiv că enumerările fac parte din valorile tip, memoria alocată pentru ele este rigidă, deci nu putem adăuga sau șterge valori din enumerare.

Valorile integer pe care le includem pentru fiecare stare din enumerare sunt constante și nu pot fi modificate în timpul execuției, doar din cod.

1. Sintaxa enumerărilor

Pentru a declara o enumerare vom folosi sintaxa următoare:

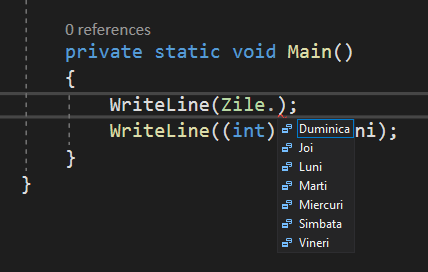
**Sintaxa:**

|  |
| --- |
| a) Nivelul de accesibilitate b) cuv cheie **enum c)numele**  {  //Conținut  } |

1. Avantajele utilizării enumerărilor și exemple

Enumerările nu sunt absolut necesare în nici un caz, dar ele oferă multe avantaje.

1. Enumerările permit stocarea unor date constante de tip int cu o denumire atașată string. Acest lucru permite să faceți diverse operații accesând doar o singură stare din enumerare.
2. Enumerările ne ajută să alegem rapid stare de care avem nevoie. Dacă creați o enumerare editorul automat va deschide un drop-down menu pentru a vă arăta toate stările disponibile.



1. Permit crearea unor verificări logice, de genul “dacă obiectul X are starea Y, execută logica dată).

**Exemplu:**

|  |
| --- |
| namespace EnumExemplu  {  class Exemplu  {  private enum Zile  {  Luni,  Marti,  Miercuri,  Joi,  Vineri,  Simbata,  Duminica  }  private static void Main()  {  Zile ziCurenta = Zile.Luni;  if(ziCurenta == Zile.Luni)  WriteLine("Incepe saptamina...");  else if (ziCurenta == Zile.Simbata || ziCurenta == Zile.Duminica) WriteLine("Weekend!");  }  }  } |

1. Enumerările sunt rapide, din motivul că acestea sunt doar numere integer cu o valoare string atașată la ele. Ceea ce le face foarte performante.

Probleme pentru rezolvare.

### **Program 1:**

1. Creați un program în care veți avea o clasă Mașină. În interiorul clasei stabiliți câteva variabile(denumire, preț etc.)
2. Creați o enumerare StareMasina, în care veți avea Nouă, Uzată, Veche, Defectată.
3. Creați o variabilă de tip StareMasina stareCurentă.
4. În clasa Execuție creați mai multe obiecte de tip Mașină cu ajutorul unui constructor cu parametri.
5. Afișați informația despre toate mașinile, cu următoarea condiție:
6. Dacă mașina este nouă afișați prețul complet.
7. Dacă mașina este Uzată afișați 70% din preț.
8. Dacă mașina este Veche afișați 40% din preț.
9. Dacă mașina este Defectată afișați 10% din preț.

### **Program 2:**

1. Creați un program în care veți stoca informațiile despre angajați într-o structură angajat.
2. În interiorul structurii includeți o enumerare StatutAngajat, în care vor fi mai multe statuturi(Activ, Concediat, Vacanta, Carantina etc.).
3. În clasa Execuție, în metode Main folosiți un constructor cu parametri pentru a crea mai mulți angajați.
4. Dacă angajatul este Activ sau în Vacanță afișați la consolă:“Angajatul X, va primi salariul de Y lei pe data de ZZZ.”.
5. Dacă angajatul este concediat afișați la consolă:“Angajatul X a fost concediat și nu va primi salariul.”
6. Dacă angajatul este în Carantina afișați :“Angajatul X este în șomaj tehnic și va primi doar (salariu\*0,8) lei.”
7. Tipul standard enum

O enumerare de tip standard va avea mai multe stări care vor primi automat valorile 0,1,2… în dependență de numărul total de stări, după cum am văzut anterior.

Dar dacă există necesitatea aceste valori pot fi modificare, programatorul poate prestabili valoarea int a fiecărei stări.

**Exemplu:**

|  |
| --- |
| namespace EnumCuParametri  {  class Exemplu  {  private enum FiguriGeometrice  {  Triunghi = 3,  Patrat = 4,  Dreptunghi = 4,  Pentagon = 5,  Hexagon = 6,  Septagon = 7,  Octagon = 8  }  private static void Main()  {  FiguriGeometrice figura1 = FiguriGeometrice.Triunghi;  FiguriGeometrice figura2 = FiguriGeometrice.Octagon;  FiguriGeometrice figura3 = FiguriGeometrice.Hexagon;  int suma = (int)figura1 + (int)figura2 + (int)figura3;  WriteLine("Numarul total de laturi pentru figurile create este:" + suma);  }  }  } |

După cum observăm în exemplul de mai sus putem crea o enumerare și defini un anumit număr pentru fiecare din elementele enumerării. În acest mod am creat o enumerare FiguriGeometrice și am definit 7 figuri și numărul acestora de laturi, ceea ce ne-a permis să calculăm numărul total de latura la 3 figuri pe care le-am creat.

1. Folosirea metodelor pentru enumerare

Enumerările au o listă de metode predefinite care pot fi utilizate pentru a îndeplini unele funcții specifice.

|  |  |
| --- | --- |
| Enum method | Description |
| CompareTo | Poate fi folosit pentru a compara valorile int care definesc o stare din enumerare.  Returnează:  **1** când prima stare are o valoare mai mare ca starea doua  **0** când stările comparate au aceeași valoare  **-1** când prima stare are o valoare mai mică ca starea doua  **Exemplu:**  namespace EnumMetode  {  class Exemplu  {  private enum FiguriGeometrice  {  Triunghi = 3,  Patrat = 4,  Dreptunghi = 4,  Pentagon = 5,  Hexagon = 6,  Septagon = 7,  Octagon = 8  }  private static void Main()  {  FiguriGeometrice figura1 = FiguriGeometrice.Triunghi;  FiguriGeometrice figura2 = FiguriGeometrice.Octagon;  WriteLine(figura1.CompareTo(figura2));  WriteLine(figura1.CompareTo(figura1));  WriteLine(figura2.CompareTo(figura1));  }  }  } |
| GetNames | Returnează toate numele stărilor din enumerare în formă de tablou string.  **Exemplu:**  class GetNames  {  enum Culori {Rosu, Verde, Albastru, Galben, Alb};  public static void Main()  {  WriteLine("Avem culorile:");  foreach (string s in Enum.GetNames(typeof (Culori)))  WriteLine(s);  }  } |
| GetValues | Similar cu metode precedentă, GetValues returnează un tablou, dar de data aceasta cu numere care reprezintă valorile int a stărilor din enumerare. |

Probleme pentru rezolvare.

### **Program 1:**

1. Creați un program în care veți avea o enumerare Obiecte cu conținutul Televizor, Calculator, Masa etc.
2. Pentru fiecare obiect setați un parametru int care va defini masa acestui obiect (Televizor = 3) de ex.
3. Creați două tablouri unidimensionale, primul de tip string, al doilea int.
4. Cu ajutorul metodei GetNames și GetValues scoateți toate valorile din enumerare și stocați-le în tablourile string și int.
5. Creați o metodă care va primi ca parametru un număr int și va returna denumirea și masa obiectului respectiv (de ex. obiectul 0 - “Obiectul 0 este Televizor și are masa de 3 kg.”

#### Tipuri nullable

1. Ce este un tip nullable? Scopurile folosirii

După cum cunoașteți deja în C# o variabilă de tip valoare nu poate primi valoarea null(veți primi o eroare în editor).

Cu ajutorul tipului nullable programatorii pot declara o variabilă care poate primi ca valoare null, ceea ce devine util atunci când logica programului presupune că va trebui să lucrăm cu valori null. În mod normal o variabilă null sau una nedeclarată generează o excepție care stopează complet programul.

**Sintaxa:**

|  |
| --- |
| //Eroare la compilare  int j = null;  //Tip nullable varianta lunga  Nullable<int> j = null;  //Tip nullable varianta scurta  int? j = null; |

Cel mai frecvent tipurile nullable se folosesc pentru:

1. Evitarea excepțiilor
2. Pentru a transmite valori null către alte metode, clase sau baze de date
3. Pentru a reprezenta variabilele nedefinite

**Exemplu:**

|  |
| --- |
| namespace Nullable  {  class Exemplu  {  //int j = null;  static Nullable<int> j = null;  private static void Main()  {  WriteLine(5);  WriteLine(j);  WriteLine(3);  }  }  } |

1. Operații specifice tipului nullable

Tipul nullable ne oferă câteva operații specifice care pot fi executate doar între variabilele de acest tip:

1. **Value** - returnează valoarea variabilei, dacă este null returnează Excepție.
2. **HasValue** - permite să verificăm dacă variabila are valoare stabilită sau null(dacă are valoare primim true, dacă este null primim false).
3. **GetValueOrDefault**  - returnează valoarea variabilei dacă este stabilită, dacă nu returnează valoarea default(0 pentru integer).

În rest pot fi utilizate aceleași operații ca și tipul de bază al variabilei.

1. Utilizare

**Exemplu:**

|  |
| --- |
| namespace Nullable  {  class Exemplu  {  static int a;  static int b;  private static void Main()  {  Metoda(a, b);  }  private static Nullable<int> Metoda(Nullable<int> \_a, Nullable<int> \_b)  {  return \_a + \_b;  }  }  } |

În exemplul de mai am scris un program care execută o operație trivială de sumare a două numere integer. Dar spre deosebire de ceea ce făceam de obicei am făcut metoda de tip Nullable<int> ceea ce ne permite să executăm programul deși variabilele a și b nu au fost inițializate și au valoarea null.

Probleme pentru rezolvare.

### **Program 1:**

1. Creați un program în care veți avea o clasă Input cu o variabilă nullable<int> input.
2. Creați un constructor cu parametri în interiorul clasei care va seta această variabilă.
3. În clasa Execuție, creați câteva obiecte de tip Input, cu ajutorul constructorului. În unele setați numere, în altele valori null.
4. Verificați dacă valoarea este null cu ajutorul metodei HasValue.
5. Dacă este null afișați “Operatia nu este posibila.”.
6. Dacă este o cifră validă multiplicați-o cu 3.