### Modulul 12. Tipuri și colecții generice în C#.

#### Tipuri generice

1. Ce este un tip generic?

Tipurile generice sunt similare cu template-urile din C++.

Generic este o clasă care permite programatorului să declare variabile și obiecte de un tip nedefinit, ceea ce vă permită să modificați tipul datelor în timpul execuției programului.

Tipurile generice în esență reprezintă placeholdere, utilitatea lor este faptul că ele pot fi transmise ca parametru către o funcție. O funcție care primește ca parametru un tip generic poate primi practic orice tip de date.

Tipurile generice se notează cu o literă cheie (de obicei **T, dar poate fi oricare alta).**

1. Avantajele utilizării.

Tipurile generice oferă un număr mare de avantaje dar cele de bază sunt:

* **Reutilizarea**. Din motiv că tipurile generice pot primi orice tip de date, acest lucru ne permite să scriem clase foarte abstracte care pot prelucra tipuri de date multiple ceea ce semnifică că nu trebuie să scriem repetat aceeasi logică.

**De exemplu**, o metodă care găsește numărul maxim într-un tablou. În mod normal am predefini care este tipul datelor din tablou(int, float, double etc), dar în acest caz limităm mult funcționalitatea acestei metode. În schimb putem folosi o metodă care primește parametri generici și calculează maximul din numerele primite, indiferent de tipul lor.

* **Performanță**. Tipurile generice beneficiază de o performanță sporită pentru că acestea nu au nevoie de boxing/unboxing.

1. Crearea claselor generice

Pentru a crea o clasă generică în C# folosim sintaxa:

|  |
| --- |
| a) Nivelul de accesibilitate b) cuv cheie class c)denumirea d)lit cheie T  public class TestClass<T>  {  } |

*Sintaxa declarării unei clase generice*

În continuare în clasa dată putem declara o variabilă/metodă/indexator/proprietate de același tip generic.

Iar la crearea unui obiecte de tipul clasei date vom defini deja tipul de date care îl transmitem, sau în cazul în care vrem să păstrăm abstractizarea putem alege tipul de date **object ,** care ne va permite să transmitem orice tip de date.

Exemplu:

|  |
| --- |
| namespace ClasaGenerica  {  public class TestClass<T>  {  T[] obj = new T[5];  int count = 0;  public void Add(T item)  {  if (count + 1 < 6)  {  obj[count] = item;  }  count++;  }  public T this[int index]  {  get { return obj[index]; }  set { obj[index] = value; }  }  }  class Program  {  static void Main(string[] args)  {  TestClass<object> intObj = new TestClass<object>();  intObj.Add(1);  intObj.Add("string");  intObj.Add(3);  intObj.Add('a');  intObj.Add(5);  for (int i = 0; i < 5; i++)  Console.WriteLine(intObj[i]);  Console.ReadKey();  }  }  } |

În exemplul de mai sus am creat o clasă generică TestClass care ne permite să stocăm 5 valori generice în ea, de asemenea putem adăuga elemente noi de tipul T în clasă prin metoda Add și putem accesa elementele din clasa dată prin intermediul indexatorului.

În clasa Program, creăm un obiect nou de tipul test class și definim tipul T ca object, ca să putem transmite orice tip de date în acesta, în rezultat vom adăuga câteva numere, un string și un caracter ceea este posibil din motiv că tipul de date este object.

Vom afișa rezultatul și vom obține toate valorile introduse.

Probleme pentru rezolvare.

### **Program 1:**

1. Folosiți exemplul de mai sus pentru a transmite în tablou mai multe valori numerice de mai multe tipuri(int, float, double, byte etc.)
2. Calculați maximul și minimul acestor valori.
3. Clase generice cu mai multe tipuri

În exemplul de mai sus am studiat clasele generice cu un singur tip, dar cu același succes putem defini o clasă cu mai multe tipuri de date.

Exemplu:

|  |
| --- |
| namespace ClasaGenerica2  {  public class TestClass<T, U, V>  {  object[] obj = new object[5];  int count = 0;  public void Add(T item)  {  if (count + 1 < 6)  {  obj[count] = item;  }  count++;  }  public void Add(U item)  {  if (count + 1 < 6)  {  obj[count] = item;  }  count++;  }  public void Add(V item)  {  if (count + 1 < 6)  {  obj[count] = item;  }  count++;  }  public object this[int index]  {  get { return obj[index]; }  set { obj[index] = value; }  }  }  class Program  {  static void Main(string[] args)  {  TestClass<float, double, string> intObj = new TestClass<float, double, string>();  intObj.Add(1);  intObj.Add("string");  intObj.Add(3);  intObj.Add('a');  intObj.Add(5);  }  }  } |

În exemplul de mai sus am creat o clasă generică TestClass care ne permite să stocăm 5 valori generice în ea, de asemenea putem adăuga elemente noi de tipul T în clasă prin metoda Add și putem accesa elementele din clasa dată prin intermediul indexatorului.

Probleme pentru rezolvare.

### **Program 1:**

1. Pe baza exemplului de mai sus creați o clasă generică care primește doar stringuri și caractere și le concatinează.
2. Metode generice

Într-o manieră similară putem folosi și metodele generice.

Pentru a declara o metodă generică vom folosi sintaxa.

**Sintaxa:**

|  |
| --- |
| a) Nivel de acces b)tipul returnat c)numele metodei **<T>** (parametri)  public void Metoda1<T>(T \_a)  {  Console.WriteLine(\_a);  } |

În esență metodele generice beneficiază de aceleași avantaje ca și clasele generice, acestea oferă un nivel înalt de abstractizare, permit reutilizarea logicii pentru tipuri multiple etc.

Exemplu:

|  |
| --- |
| namespace MetodaGenerica  {  class Program  {  static void Inlocuire<T>(ref T a, ref T b)  {  T temp;  temp = a;  a = b;  b = temp;  }  static void Main(string[] args)  {  int a = 10, b = 120;  WriteLine("Inainte de inlocuire: {0}, {1}", a, b);  Inlocuire<int>(ref a, ref b);  WriteLine("Dupa inlocuire: {0}, {1}", a, b);  }  }  } |

În programul de mai sus am folosit o metodă generică pentru a înlocui valorile a două variabile, și din motiv că acestea sunt de tip ref, ele au fost modificate și în metoda Main().

Probleme pentru rezolvare.

### **Program 1:**

1. Pe baza exemplului de mai sus creați un program care conține o metodă generică cu 3 parametri.
2. Dacă parametrul este de tip număr faceți ridicarea la patrat, dacă este de tip string afișați primele 3 caractere. Afișați rezultatul la consolă

#### Iteratori

1. Ce este un iterator?

Un iterator în C# este un metodă ciclică care ne permite să accesăm fiecare element dintr-un array, listă sau alte forme de colecții.

Un iterator este în esență un ciclu **foreach** care returnează o colecție de elemente cu ajutorul cuvintelor cheie **yield return.**

1. Sintaxa declarării

Un iterator se declară cu ajutorul cuvântului cheie **IEnumerable<tip date>** și cere necesar tipul datelor care le întoarce.

**Sintaxa:**

|  |
| --- |
| **a)IEnumerable b)<string>** **c)Nume()**  {  foreach (**//o colectie de date)**  yield return **//un element individual de tip string(în cazul dat)**  } |

**Exemplu:**

|  |
| --- |
| namespace Iterator  {  class IteratorExample  {  public static IEnumerable<string> GetArray()  {  int[] arr = new int[] { 5, 8, 6, 9, 1 };  foreach (var element in arr)  yield return element.ToString()+"A";  }  public static void Main(string[] args)  {  IEnumerable<string> elements = GetArray();  foreach (var element in elements)  WriteLine(element);  }  }  } |

1. Utilitatea și exemple

Utilitatea iteratorilor este faptul că aceștia vă permit să convertiți un număr de variabile(dintr-un masiv sau individuale) într-o colecție tipizată. Acest lucru are multe avantaje pentru că după aceea puteți folosi metodele din librăria LINQ pentru a procesa această colecție.

#### Colecțiile în C#

1. Ce este o colecție?

Colecțiile în C# sunt clase specializate pentru stocarea și accesarea datelor. Aceste clase oferă o gamă largă de metode și proprietăți create intenționat cu scopul de a ajuta la procesarea volumelor mari de informație.

1. Colecții generice

Colecțiile generice fac parte din clasa System.Collections.Generic și acestea permit programatorilor să trimită date generice către o colecție pentru utilizare ulterioară.

**Exemplu:**

|  |
| --- |
| namespace ColectiiGenerice  {  class ColectieGenerica  {  public static void Main(string[] args)  {  ArrayList arList = new ArrayList();  arList.Add(1);  arList.Add("Two");  arList.Add(true);  arList.Add(100.45);  arList.Add(DateTime.Now);  foreach (var data in arList)  WriteLine(data.ToString());  }  }  } |

Să considerăm exemplul dat, am utilizat un ArrayList care este o colecție generică pentru a introduce în ea date de tipul int, sting, bool, decimal, DateTime.

În esență colecțiile generice reprezintă containerul perfect pentru stocarea datelor aleatorii în C#, dar trebuie să implementați măsuri de siguranță în cazul în care veți avea nevoie să utilizați datele din interiorul acestora.

Probleme pentru rezolvare.

### **Program 1:**

1. Folosiți exemplul de mai sus, trimiteți date adiționale în ArrayList, dar afișați doar datele de tip string.

Probleme pentru rezolvare.

### **Program 2:**

1. Folosiți exemplul de mai sus, trimiteți date adiționale în ArrayList, pentru fiecare variabilă de tip int afișați un mesaj la consolă.
2. Tipurile de colecții (Liste, Dicționare etc.)
3. Liste

Listele reprezintă o clasă specializată în C# primește un singur parametru care reprezintă tipul datelor care pot fi stocate în interiorul listei. Avantajele utilizării listelor în loc de tablouri este faptul că acestea își pot modifica oricând dimensiunea(putem adăuga și scoate valori din listă) și putem utiliza metode specializate pentru prelucrarea elementelor din listă.

**Sintaxa:**

|  |
| --- |
| List<int> intList = new List<int>(); |

**Proprietățile listelor:**

|  |  |
| --- | --- |
| Proprietate | Utilizare |
| List[X] | Permite returnarea sau motificare unui element din listă(exact ca si array) |
| Count | Returnează lungimea listei(silimar cu array Length) |

**Metodele listelor:**

|  |  |
| --- | --- |
| Metode | Utilizare |
| Add | Adaugă un element la sfârșitul listei |
| AddRange | Adaugă elementele dintr-un tablou sau colecție la sfârșitul listei(dacă tipurile se potrivesc) |
| Clear | Șterge toate elementele din listă |
| Contains | Verifică dacă un element X este prezent în listă(returnează true sau false) |
| Find | Găsește primul element care se potrivește condiției indicate(condiți este o expresie Lambda) |
| Insert | Inserează un element pe poziția indicată(celelalte elemente sub deplasate) |
| InsertRange | Inserează mai multe elemente din altă colecție pe poziția indicată |
| Remove | Șterge primul element care se potrivește cu parametrul indicat |
| RemoveAt | Șterge elementul de pe poziția indicată |
| RemoveRange | Șterge toate elementele care se potrivesc cu condiția indicată |
| Sort | Sortează toate elementele |
| TrimExcess | Șterge spațiile libere din listă pentru a realoca memoria(în cazul în care unele elemente din listă au fost șterse) |
| TrueForAll | Determină dacă toate elementele din listă corespun condiției indicate. |

**Exemplu:**

|  |
| --- |
| namespace Liste  {  class ColectieGenerica  {  public static void Main(string[] args)  {  List<int> intList = new List<int>();  intList.Add(5);  foreach (var numar in intList)  WriteLine(numar);  }  }  } |

Probleme pentru rezolvare.

### **Program 1:**

1. Creați un program în care veți defini o listă de caractere.
2. Creați un tablou de caractere și folosiți funcția InsertRange pentru a-l adăuga la mijlocul listei.
3. Verificați dacă lista conține caracterul Z.
4. Ștergeți toate litere ‘a’ din listă.
5. Sortați lista.
6. Afișați rezultatul.
7. Dicționare

Dicționarele reprezintă colecții C# care primesc doi parametri, primul Key care definește cheia care definește datele, și al doilea - Value care denotă valoarea înregistrării.

Dicționarele sunt foarte asemănătoare cu o înregistrate într-o bază de date.

Când vorbim de dicționare în C# definim fiecare valoare din dicționar ca o pereche, care conține valoare și cheie.

**Observație:** Dicționarele nu pot conține două chei similare.

**Proprietățile listelor:**

|  |  |
| --- | --- |
| Proprietate | Utilizare |
| Count | Returnează numărul perechilor din dicționar. |
| Dicționar[x, y] | Setează sau primește valoarea cu cheia X și valoarea Y |
| Keys | Returnează cheile tuturor elementelor ca o nouă colecție |
| Values | Returnează valorile tuturor elementelor ca o nouă colecție |

**Metodele listelor:**

|  |  |
| --- | --- |
| Metode | Utilizare |
| Add | Adaugă un element(trebuie trimise ca parametru cheia și valoarea) |
| Remove | Șterge primul element care coincide cu parametrii introduși(putem introduce doar cheia, sau cheia împreună cu valoarea) |
| ContainsKey | Verifică dacă dicționarul conține cheia X |
| ContainsValue | Verifică dacă dicționarul conține parametrul X |
| Clear | Șterge toate înregistrările din dicționar |
| TryGetValue | Dacă identifică elementul cu cheia X returnează valoarea dacă nu returnează false. |
| TrimExcess | Șterge spațiile libere |

**Exemplu:**

|  |
| --- |
| namespace Dictionare  {  class Dictonar  {  public static void Main(string[] args)  {  Dictionary<int, string> dictionar = new Dictionary<int, string>();  dictionar.Add(1, "este");  dictionar.Add(101, "Doggo");  dictionar.Add(102, "Doggo");  dictionar.Remove(2);  dictionar.Add(5, "Cuvint");  dictionar.Add(6, "Cuvint2");  foreach (var number in dictionar.Keys)  WriteLine(number);  WriteLine("Dictionarul contine cheia:"+dictionar.ContainsKey(5));  WriteLine("Dictionarul contine valoarea:"+dictionar.ContainsValue("este"));  dictionar.Clear();  dictionar.TrimExcess();  foreach (var valoare in dictionar)  WriteLine(valoare);  }  }  } |

În exemplul de mai sus am folosit un dicționar cu date de tip int și string și am executat mai multe operații asupra acestora cu ajutorul metodelor indicate mai sus.

Probleme pentru rezolvare.

### **Program 1:**

1. Creați un program în care veți defini un dicționar cu cheia de tip char, și valoarea de tip număr.
2. Cu ajutorul unui ciclu for adăugați 10 variabile aleatorii pentru cheie și valoarea dicționarului.
3. Verificați dacă dicționarul conține cheia z.
4. Dacă da - printați toate cheile din dicționar.
5. Dacă nu - printați toate valorile.
6. SortedList

SortedList sunt colecții similare cu Dicționarele (practic aceleași metode și proprietăți) cu excepția faptului că de fiecare dată când adăugați elemente în sorted list, ele se sortează automat.

**Exemplu:**

|  |
| --- |
| namespace SortedList  {  class Dictonar  {  public static void Main()  {  SortedList<int, string> sortedList = new SortedList<int, string>  {  {3, "Three"},  {4, "Four"},  {1, "One"},  {5, "Five"},  {2, "Two"}  };  foreach (var \_var in sortedList)  WriteLine(\_var);  }  }  }  Rezultat:  [1, One]  [2, Two]  [3, Three]  [4, Four]  [5, Five] |

1. ArrayList

ArrayList este un tip de colecție în C# care poate primi orice tip de date.

**Exemplu:**

|  |
| --- |
| namespace ArrayLista  {  class Dictonar  {  private static void Main()  {  ArrayList myArrayList = new ArrayList();  myArrayList.Add(100);  myArrayList.Add('c');  myArrayList.Add(3.6001);  myArrayList.Add("Two");  myArrayList.Add(4.5f);  myArrayList.Add(myArrayList);  foreach (var \_var in myArrayList)  WriteLine(\_var);  WriteLine("\n");  myArrayList.Reverse();  foreach (var \_var in myArrayList)  WriteLine(\_var);  }  }  } |

1. Stack

<https://www.tutorialsteacher.com/csharp/csharp-stack>

1. Queue

<https://www.tutorialsteacher.com/csharp/csharp-queue>

Probleme pentru rezolvare.

### **Program 1:**

1. Pe baza exemplului de mai sus(pagina 3, namespace ClasaGenerica2) creați o clasă generică care primește doar stringuri și caractere și le concatinează.

### **Program 2:**

1. Creați un program în care veți defini un dicționar cu cheia de tip char, și valoarea de tip număr.
2. Cu ajutorul unui ciclu for adăugați 10 variabile aleatorii pentru cheie și valoarea dicționarului.
3. Verificați dacă dicționarul conține cheia z.
4. Dacă da - printați toate cheile din dicționar.
5. Dacă nu - printați toate valorile.

### **Program 3:**

1. Folosiți exemplul de mai sus(pagina 6, namespace Colectii Generice), trimiteți date adiționale în ArrayList, pentru fiecare variabilă de tip int afișați un mesaj la consolă.

### **Program 4:**

1. Creați un program în care veți defini un dicționar cu cheia de tip char, și valoarea de tip număr.
2. Cu ajutorul unui ciclu for adăugați 10 variabile aleatorii pentru cheie și valoarea dicționarului.
3. Verificați dacă dicționarul conține cheia z.
4. Dacă da - printați toate cheile din dicționar.
5. Dacă nu - printați toate valorile.