Colectii, comparatii, conversii

Ne propunem sa abordam urmatoarele probleme: cum se definesc colectiile, care sunt principalele tipuri colectii, cum se compara diverse tipuri si cum se utilizeaza operatorul is, cum se definesc si cum se utilizeaza conversiile, cum se utilizeaza operatorul as.

Colectiile permit mentinerea unui grup de obiecte. Contrar tablourilor, colectiile permit realizarea unor operatii mai avansate precum controlul obiectelor pe care acestea le contin, cautarea si sortarea intr-o colectie.

Comparatii. Lucrand cu obiecte, de multe ori este necesar a le compara. Acest fapt este important atunci cand dorim sa realizam o operatie de sortare. Vom analiza modul in care putem compara doua obiecte, inclusiv supraincarcarea operatorilor, si cum se utilizeaza interfetele IComparable si IComparer.

Conversii. In multe situatii am utitlizat un cast pentru ca un obiect de un anumit tip sa fie privit ca obiect de un alt tip. Vom analiza in detaliu aceasta operatie.

Colectii. In cursurile precedente am utilizat tablourile (clasa Array din spatiul de nume System) pentru a crea tipuri de variabile care contin un numar de obiecte. Tablourile au limitele sale. Una dintre aceste limite este aceea ca tablourile au o dimensiune fixata. Nu putem adauga noi elemente intr-un tablou.

Tablourile, implementate ca instante ale clasei Array reprezinta doar unul din tipurile cunoscute drept colectii. Colectiile sunt utilizate pentru a mentine o lista de obiecte si, in general, au o functionalitate mai mare decat tablourile. O mare parte a acestei functionalitati este asigurata prin implementarea interfetelor continute de spatiul de nume System.Collections. Acest spatiu de nume contine o serie de clase care implementeaza aceste interfete.

Intrucat *functionalitatea unei colectii* (inclusiv accesarea elementelor colectiei utilizand un index) *este asigurata prin interfete*, nu suntem obligati a ne limita doar la utilizarea clasei Array. Mai degraba, putem crea propriile noastre colecti. Un avantaj este acela ca putem crea colectii puternic tipizate (strongly typed). Aceasta inseamna ca atunci cand extragem un element dintr-o colectie de acest tip nu este necesar sa utilizam un cast pentru a obtine tipul corect al elementului. Un alt avantaj este capacitatea de a expune metode specializate.

Cateva dintre intervetele spatiului de nume System.Collections, care ofera functionalitatea de baza, sunt:

IEnumerable ofera capacitatea de a cicla intr-o colectie prin intermediul metodei

GetEnumerator();

ICollection ofera capacitatea de a obtine numarul de obiecte continute de o colectie si posibilitatea de a copia elementele intr-un tablou. Mosteneste interfata IEnumerable.

IList ofera o lista a elementelor unei colectii impreuna cu capacitatea de a accesa aceste elemente. Mosteneste interfetele IEnumerable si ICollection.

IDictionary similar interfetei IList, insa ofera o lista de elemente accesibile prin intermediul unui cuvant cheie (key value) si nu un index. Mosteneste interfetele IEnumerable si ICollection.

Classa Array implementeaza primele trei interfete, insa nu suporta cateva trasaturi avansate ale interfetei IList, si reprezinta o lista cu un numar fix de elemente.

Utilizarea colectiilor. Clasa ArrayList din spatiul de nume System.Collections, la randul ei implementeaza interfetele IEnumerable, ICollection si IList, insa o face intr-un alt mod decat clasa Array. Contrar clasei Array, clasa ArrayList poate fi utilizata pentru a crea o lista cu un numar variabil de elemente.

Aplicatia de mai jos exemplifica modul in care se poate utiliza o serie de metode si proprietati implementate de ArrayList.

Exemplu: clasa ArrayList

```
using System;  using System.Collections.Generic;
using System.Collections; using System.Linq;   using System.Text;
namespace ExempluArrayList
{
    public enum TipCasa  {  Paie = 0,  Lemn = 1, Piatra = 2}
    public abstract class Animal
   {  protected string nume;
      public string Nume
      { get { return nume; }     set { nume = value; }        }
      public Animal()
      {   nume = "Animalul nu are nume“;  }
      public Animal(string numeNou)
      {   nume = numeNou;  }
      public void Hrana()
      {     Console.WriteLine("{0} a mancat.", nume);   }
   }
```

```
public class Porc : ExempluArrayList.Animal
    {
        public TipCasa Material;
        public void Casa()
        {  Console.WriteLine("{0} are o casa din {1}", nume, Material);      }

        public Porc(string numeNou, TipCasa material)
            : base(numeNou)
        {    Material = material;    }
    }

public class Gasca : ExempluArrayList.Animal
    {  public Gasca(string numeNou):base(numeNou)
        {       }
            public void FaceOua()
        {
            Console.WriteLine("{0} s-a ouat.", nume);
        }
    }
```

```
class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Creati un tablou de obiecte de tip Animal:");
            Animal[] animalArray = new Animal[2];
            Porc porculNr1 = new Porc("Nif-Nif", TipCasa.Paie);
            animalArray[0] = porculNr1;
            animalArray[1] = new Gasca("Amelia");
            foreach (Animal animalulMeu in animalArray)
            {
                Console.WriteLine("Un nou obiect de tipul {0} a fost adaugat in tablou,
        Nume = {1}",   animalulMeu.ToString(), animalulMeu.Nume);
            }
            Console.WriteLine("Tabloul contine {0} obiecte.“,   animalArray.Length);
            animalArray[0].Hrana();
            ((Gasca)animalArray[1]).FaceOua();
            Console.WriteLine();
```

```
Console.WriteLine("Creati un ArrayList de obiecte de tip Animal:");
    ArrayList animalArrayList = new ArrayList();
    Porc porculNr2 = new Porc("Nuf-Nuf", TipCasa.Lemn);
    animalArrayList.Add(porculNr2);
    animalArrayList.Add(new Porc("Naf-Naf",TipCasa.Piatra));
    animalArrayList.Add(new Gasca("Abigail"));
    foreach (Animal animalulMeu in animalArrayList)
    {
        Console.WriteLine("Un nou obiect de tipul {0} a fost adaugat in colectie,
              Nume = {1}“,  animalulMeu.ToString(), animalulMeu.Nume);
     }
    Console.WriteLine("Colectia ArrayList contine {0} obiecte.“,
                                                    animalArrayList.Count);
    ((Animal)animalArrayList[0]).Hrana();
    ((Porc)animalArrayList[1]).Casa();
    ((Gasca)animalArrayList[2]).FaceOua();
    Console.WriteLine();
```

```
        Console.WriteLine("Alte operatii in ArrayList:");
            animalArrayList.RemoveAt(1);
            ((Animal)animalArrayList[1]).Hrana();
            animalArrayList.AddRange(animalArray); //metoda virtuala

            ((Porc)animalArrayList[2]).Casa();

            Console.WriteLine("Animalul cu numele {0} are indexul {1}.",
            porculNr1.Nume, animalArrayList.IndexOf(porculNr1));

            Console.ReadKey();
        }
    }
}
```

Cum se definesc colectiile. In aplicatia precedenta, am fi putut adauga colectiei animalArrayList obiecte de orice tip. Normal ar fi ca sa putem adauga doar elemente de un anumit tip (in exemplul anterior, tipul Animal). Este momentul sa vedem cum putem defini colectii puternic tipizate (strongly typed).

O modalitate de a defini o colectie puternic tipizata este de a implementa manual metodele necesare. Insa acesta este un proces complex care necesita timp.

O alta optiune este aceea de a utiliza clasa abstracta CollectionBase din spatiul de nume System.Collections drept clasa de baza pentru colectia puternic tipizata care urmeaza a fi definita. Clasa CollectionBase expune interfetele IEnumerable, ICollection si IList, insa ofera doar o mica parte din implementarea necesara, de notat metodele Clear() si RemoveAt() ale interfetei IList si proprietatea Count a interfetei ICollection. Daca utilizatorul doreste o alta functionalitate atunci trebuie sa o implementeze el insusi.

Pentru a facilita aceasta implementare, CollectionBase ofera doua proprietati protejate care permit accesul la obiectele stocate. Se poate utiliza proprietatea List, care permite accesarea elementelor prin interfetei IList si InnerList, un obiect ArrayList utilizat pentru stocarea elementelor.

Spre exemplu, colectia care ar urma sa stocheze obiecte de tip Animal ar putea fi definita astfel:

```
public class Animalee : CollectionBase
{
  public void Add(Animal unNouAnimal)
            {  List.Add(unNouAnimal); }
  public void Remove(Animal animalulX)
            { List.Remove(animalulX);}
  public Animalee()  {  }
}
```

Aici, Add() si Remove() au fost implementate ca metode strongly typed, prin utilizarea metodelor standard Add() si Remove() ale interfetei IList pentru a accesa elementele colectiei. Metodele expuse vor merge doar pentru clasa Animal sau clase derivate din Animal, spre deosebire de implementarea clasei ArrayList pentru care putem adauga sau extrage orice obiect.

Clasa CollectionBase faciliteaza utilizarea buclei foreach. Astfel puteti utiliza spre exemplu:

```
    Animalee colectieAnimale = new Animalee();
    colectieAnimale.Add(new Gasca(" Daffy Duck "));
    foreach (Animal animalulMeu in colectieAnimale)
  {
            Console.WriteLine ("Un nou obiect de tipul {0} a fost adaugat in colectie,
            Nume = {1}“,  animalulMeu.ToString(), animalulMeu.Nume);
  }
```

Nu este permisa accesarea elementelor colectiei prin intermediul unui index. Adica nu putem scrie:

```
colectieAnimale[0].Hrana();
```

pentru a accesa primul element din colectie, cu exceptia cazului in care colectia contine o indexare. Pentru clasa Animalee o indexare poate fi definita astfel:

```
public class Animalee : CollectionBase
{
...
public Animal this[int indexulAnimalului]
{
get { return (Animal)List[indexulAnimalului]; }
set { List[indexulAnimalului] = value; }
}
}
```

Codul indexarii de mai sus utilizeaza indexarea pusa la dispozitie de intefata IList prin intermediul proprietatii List, adica: (Animal)List[indexulAnimalului]; Castul este necesar deoarece indexarea pusa la dispozitie de IList returneaza un obiect de tipul object.

De notat ca indexarea de mai este de tipul Animal. Astfel, putem scrie

```
colectieAnimale[0].Hrana();
```

spre deosebire de cazul anterior (vezi aplicatia), unde era utilizata clasa ArrayList, care necesita utilizarea castului, spre exemplu ((Porc)animalArrayList[2]).Casa();

Drept exemplu, rescrieti aplicatia precedenta in care adaugati clasa Animalee si inlocuiti blocul metodei Main() , asa cum se specifica in urmatoarele doua slide-uri.

Adaugati clasa de mai jos:

```
public class Animalee:CollectionBase
{
    public void Add(Animal unNouAnimal)
     { List.Add(unNouAnimal);}
public void Remove(Animal animalulX)
     {List.Remove(animalulX);}
public Animalee()
{
}
public Animal this[int indexulAnimalului]
{
get
{return (Animal)List[indexulAnimalului];}
set
{List[indexulAnimalului] = value;}
}
}
```

Inlocuiti blocul metodei Main() cu cel de mai jos:

```
{       Console.WriteLine("Creati o colectie de Animalee de tip Animal:");
    Animalee animale = new Animalee();
    Porc porculNr2 = new Porc("Nuf-Nuf", TipCasa.Lemn);
    animale.Add(porculNr2);
    animale.Add(new Porc("Naf-Naf",TipCasa.Piatra));
    animale.Add(new Gasca("Abigail"));
    foreach (Animal animalulMeu in animale)
    {
       Console.WriteLine("Un nou obiect de tipul {0} a fost adaugat in colectie, Nume =
{1}",  animalulMeu.ToString(), animalulMeu.Nume);
            }
    Console.WriteLine("Colectia contine {0} obiecte.", animale.Count);
    animale[0].Hrana();
    ((Porc)animale[1]).Casa();
    ((Gasca)animale[2]).FaceOua();
    Console.WriteLine();
    Console.WriteLine("Alte operatii in ArrayList:");
    animale.RemoveAt(1);
    animale[1].Hrana();
    animale.Remove(porculNr2);
    Console.WriteLine("Au ramas {0} animale.", animale.Count.ToString());
    Console.ReadKey();
}
```

Colectii care utilizeaza cuvinte cheie. Interfata IDictionary. In locul interfetei IList, este posibil ca o colectie sa implementze in mod similar interfata IDictionary, care permite accesarea elementelor prin intermediul unor cuvinte cheie (precum un string), in locul unui index.

La fel ca in cazul colectiilor indexate, exista clasa DictionaryBase care poate fi utilizata pentru a simplifica implementarea interfetei IDictionary. Clasa DictionaryBase implementeaza, de asemenea, IEnumerable si ICollection oferind astfel facilitatile de baza pentru manipularea colectiilor.

La fel ca si CollectionBase, clasa abstracta DictionaryBase implementeaza o serie de membrii obtinuti prin intefetele suportate de aceasta clasa. Membrii Clear() si Count sunt implementati, insa RemoveAt() nu, intrucat aceasta metoda nu este continuta de interfata IDictionary.

Codul din urmatorul slide reprezinta o versiune alternativa a clasei Animalee, insa de aceasta data, este derivata din DictionaryBase.

De remarcat ca metoda Add( ) contine doi parametrii, un cuvant cheie si un obiect. Aceasta deoarece DictionaryBase contine proprietatea Dictionary care are ca tip interfata IDictionary. Aceasta interfata are metoda sa Add() care are doi parametrii (doua obiecte de tipul object), primul un cuvant cheie (a key) si cel de-al doilea un element de stocat in colectie. De asemenea, Remove() are ca parametru cuvantul cheie, aceasta inseamna ca elementul avand cuvantul cheie specificat va fi sters din colectie, iar indexarea utilizeaza un cuvant cheie de tip string si nu un index pentru a stoca elementele.

```
public class Animalee : DictionaryBase
{
public void Add(string noulID, Animal unNouAnimal)
{Dictionary.Add(noulID, unNouAnimal);}

public void Remove(string animalID)
{Dictionary.Remove(animalID);}

public Animalee()
{}
public Animal this[string animalID]
{
get { return (Animal)Dictionary[animalID];}
set { Dictionary[animalID] = value;}
}
}
```

O diferenta importanta intre colectiile bazate pe DictionaryBase si respectiv CollectionBase este faptul ca bucla foreach functioneaza oarecum diferit. Colectiile din exemplele anterioare permiteau extragerea obiectelor direct din colectie.

Utilizand foreach cu clasele derivate din DictionaryBase obtinem o structura DictionaryEntry, un alt tip definit in System.Collection. Pentru a obtine un obiect de tip Animal, trebuie sa utilizam membrul Value al acestei structuri, sau daca dorim sa obtinem cuvantul cheie utilizam membrul Key.

Astfel daca dorim sa obtinem cod echivalent cu cel de mai jos (utilizat in exemplele anterioare, unde animale reprezenta un obiect de de tipul unei clase derivate din CollectionBase):

```
foreach (Animal animalulMeu in animale)
{   Console.WriteLine("Un nou obiect de tipul {0} a fost adaugat in colectie, Nume = {1}",  animalulMeu.ToString(), animalulMeu.Nume);   }
```

atunci avem nevoie de urmatoarele instructiuni daca animale reprezenta acum un obiect de de tipul unei clase derivate din DictionaryBase):

```
foreach (DictionaryEntry myEntry in animale)
{
Console.WriteLine("Un nou obiect de tipul {0} a fost adaugat in colectie, Nume = {1}", myEntry.Value.ToString(), ((Animal) myEntry.Value).Nume);
  }
```

Exercitiu: Rescrieti programul anterior folosind clasa Animalee derivata din DictionaryBase.

Comparatii. De multe ori in practica avem de *comparat doua sau mai multe obiecte*. Spre exemplu, atunci cand dorim sa *sortam sau sa cautam* intr-o colectie. Practic, realizam doua tipuri de comparatii: comparatii pentru determinarea tipurilor obiectelor *(type comparisons)* si comparatii intre valorile obiectelor *(value comparisons).*

Type comparisons. Atunci cand dorim sa comparam doua obiecte trebuie sa cunoastem mai intai tipul obiectelor. Aceasta informatie ne permite sa determinam daca este posibila o comparatie a valorilor acestor obiecte.

O prima posibilitate de a determina tipul unui obiect este aceea de a utiliza metoda GetType(), pe care toate clasele o mostenesc de la clasa object, in combinatie cu operatorul typeof(). Spre exemplu:

```
if (obiectulMeu.GetType() = = typeof(ClasaMea))
{
        //obiectulMeu este instanta a clasei ClasaMea
}
```

Metoda GetType() returneaza tipul obiectului obiectulMeu in forma System.Type (un obiect de tipul acestei clase), in timp ce operatorul typeof() converteste numele clasei ClasaMea intr-un obiect SystemType.

De remarcat ca expresia instructiunii if se evalueaza cu true doar daca obiectulMeu este de tipul ClasaMea. Daca am considera secventa de cod de mai sus in cazul programului anterior si am lua drept obiectulMeu obiectul porculNr1 si drept ClasaMea clasa Animal (clasa de baza pentru clasa Porc) atunci expresia se evalueaza cu false.

O alta posibilitate, mult mai directa, de a determina tipul unui obiect este de a utiliza operatorul is. Acest operator permite sa determinam daca un obiect este sau poate fi convertit intr-un tip dat. In fiecare dintre aceste doua cazuri operatorul se evalueaza cu true.

Operatorul is are urmatoarea sintaxa:

<operand> is <type>

Rezultatele posibile ale acestei expresii sunt:

a) Daca <type> este o clasa atunci rezultatul este true daca <operand> este de tipul acelei clase sau de tipul unei clase derivate din clasa <type> sau poate fi impachetat in acel tip;

b) Daca <type> este o interfata atunci rezultatul este true daca <operand> este de acel tip sau daca este de un tip care implementeaza acea interfata;

c) Daca <type> este un tip valoric atunci rezultatul este true daca <operand> este de acel tip sau este de un tip care poate fi despachetat in acel tip.

De remarcat ca prima modalitate, adica cea care utilizeaza GetType(), determina doar daca un obiect este de un tip dat, in timp ce utilizand operatorul is avem mult mai multe posibilitati.

Urmatorul exemplu arata cum se poate folosi operatorul is.

```
using System;     using System.Collections.Generic;
using System.Linq;     using System.Text;
namespace Exemplu
{class Checker
{   public void Check(object param1)
{
   if (param1 is ClassA)
      Console.WriteLine("Variabila poate fi convertita in tipul ClassA.");
   else
      Console.WriteLine("Variabila NU poate fi convertita in tipul ClassA.");
   if (param1 is IInterfata)
      Console.WriteLine("Variabila poate fi convertita in IInterfata.");
   else
      Console.WriteLine("Variabila NU poate fi convertita in IInterfata.");
   if (param1 is Structura)
      Console.WriteLine("Variabila poate fi convertita in Structura.");
   else
      Console.WriteLine("Variabila NU poate fi convertita in Structura.");
}
}
```

```
interface IInterfata {}
class ClassA : IInterfata
{ }
class ClassB : IInterfata
{ }
class ClassC
{ }
class ClassD : ClassA
{ }
struct Structura : IInterfata
{ }
class Program
{    static void Main(string[] args)
  { Checker check = new Checker();
    ClassA obA = new ClassA();
    ClassB obB = new ClassB();
    ClassC obC = new ClassC();
    ClassD obD = new ClassD();
    Structura structuraS = new Structura();
    object obiect = structuraS;
Console.WriteLine("Analizam variabila
                    de tipul clasei ClassA:");
    check.Check(obA);
    Console.WriteLine("\nAnalizam variabila
                    de tipul clasei ClassB:");
    check.Check(obB);
    Console.WriteLine("\nAnalizam variabila
                    de tipul clasei ClassC:");
    check.Check(obC);
    Console.WriteLine("\nAnalizam variabila
                    de tipul clasei ClassD:");
    check.Check(obD);
    Console.WriteLine("\nAnalizam variabila
          de tipul structurii Structura:");
    check.Check(structuraS);
    Console.WriteLine("\nAnalizam variabila
      de tipul structurii Structura impachetata:");
    check.Check(obiect);
    Console.ReadKey();
}
}
}
```

Value comparisons. Sa presupunem ca avem o clasa numita Persoana care contine o proprietate de tip intreg numita Varsta. Putem compara doua obiecte de tip Persoana astfel:

```
if (persoana1.Varsta > =persoana2.Varsta)
{ ….}
```

Exista insa si alternative. Spre exemplu, in unele situatii ar fi de preferat sintaxa:

```
if (persoana1 > =persoana2)
{ ….}
```

Aceasta secventa de cod este posibila daca pentru clasa Persoana au fost supraincarcati operatorii >= si <= (a se vedea un curs anterior). Aceasta tehnica este interesanta, insa trebuie utilizata cu precautie. In codul de mai sus nu este evident ca sunt comparate varstele celor doua persoane (ar putea fi inaltimile sau masele celor doua persoane).

O alta optiune de a compara valorile obiectelor este de a utiliza interfetele IComparable si Icomparer. Aceasta tehnica este suportata de diverse clase colectii, facand-o o modalitate excelenta de sortare a obiectelor.

Interfetele IComparable si IComparer. Aceste interfete ne permit sa definim modul de comparare a obiectelor intr-un mod standard. IComparable face parte din spatiul de nume System, iar IComparer din System.Collections.

Intre cele doua interfete avem urmatoarele diferente: IComparable este implementata de clasa obiectului care urmeaza a fi comparat si permite comparatii intre acel obiect si un alt obiect; IComparer este implementata de o clasa separata si permite compararea oricaror doua obiecte.

IComparable expune metoda CompareTo(), care accepta un object ca parametru si returneza un intreg pozitiv, egal cu zero sau negativ daca obiectul instanta este mai mare, egal sau mai mic (relativ la o operatie de ordine dorita de utilizator) decat obiectul primit ca parametru. Spre exemplu, clasa Persoana ar putea implementa CompareTo() asa incat parametrul metodei sa fie de tipul clasei Persoana, iar rezultatul intors de metoda sa specifice daca persoana curenta este mai in varsta decat persoana primita ca parametru al metodei. Pentru compararea varstei s-ar putea utiliza urmatoarea secventa de cod:

```
if(persoana1.CompareTo(persoana2) = = 0)
        {Console.WriteLine("Cele doua persoane au acceasi varsta");}
else if(persoana1.CompareTo(persoana2) > 0)
        {Console.WriteLine("persoana1 este mai in varsta decat persoana2");}
else
        {Console.WriteLine("persoana1 este mai tanara decat persoana2");}
```

IComparer expune singura metoda Compare(), care accepta doua obiecte ca parametri si returneaza un intreg la fel ca metoda CompareTo(). Pentru un obiect, sa zicem comparaPersoane, care suporta interfata IComparer (sau altfel spus, obiectul comparaPersoane este instanta a unei clase care implementeaza interfata IComparer), se poate utiliza urmatoarea secventa de cod:

```
if(comparaPersoane.Compare(persoana1, persoana2) = = 0)
        {Console.WriteLine("Cele doua persoane au acceasi varsta");}
else if(comparaPersoane.Compare(persoana1, persoana2) > 0)
        {Console.WriteLine("persoana1 este mai in varsta decat persoana2");}
else
        {Console.WriteLine("persoana1 este mai tanara decat persoana2");}
```

In fiecare din cele doua cazuri, parametrii metodelor sunt de tipul object. Astfel, pentru a preveni comparea unor obiecte care nu pot fi comparate, la implementarea acestor metode, inaintea returnarii unui rezultat, trebuie sa faceti mai intai o comparare a tipurilor obiectelor si eventual lansarea unei exceptii daca obiectele nu au tipul convenit (a se vedea exemplul care sorteaza o colectie).

Arhitectura .NET include o implementare standard (default) a interfetei IComparer in cadrul clasei Comparer. Aceasta clasa face parte din System.Collections. Aceasta clasa permite comparatii specifice intre tipurile simple (inclusiv tipul string) precum si orice tip care implementeaza interfata IComparable. Spre exemplu, se poate utiliza urmatoarea secventa de cod:

```
string primulString="Primul string";
string alDoileaString= "Al doilea string";
Console.WriteLine("Daca comparam {0} cu {1} obtinem: {2}", primulString,
        alDoileaString, Comparer.Default.Compare(primulString, alDoileaString));
int primulIntreg=20;
int alDoileaIntreg= 32;
Console.WriteLine("Daca comparam {0} cu {1} obtinem: {2}", primulIntreg,
        alDoileaIntreg, Comparer.Default.Compare(primulIntreg, alDoileaIntreg));
```

Rezultatul este urmatorul:

```
Daca comparam Primul string cu Al doilea string obtinem: 1
Daca comparam 20 cu 32 obtinem: -1
```

Deoarece P este dupa A in alfabet, rezulta ca P este evaluat ca mai mare decat A si rezultatul este pozitiv. In celalalt caz, 20 este mai mic decat 32 si deci rezultatul este negativ. Rezultatul nu precizeaza magnitudinea diferentei.

A fost utilizat membrul Comparer.Default al clasei Comparer pentru a obtine o instanta a clasei Comparer. (Comparer.Default returneaza in obiect de tipul clasei Comparer).

Sortarea colectiilor utilizand interfetele IComparable si IComparer. O serie de colectii permit sortarea elementelor lor. ArrayList este un exemplu. Aceasta clasa contine metoda Sort(), care poate fi utilizata fara parametrii, caz in care sorteaza utilizand comparatorul default sau i se poate transmite ca parametru o interfata IComparer, caz in care se utilizeaza aceasta interfata pentru a compara perechi de obiecte.

Daca colectia ArrayList contine numai tipuri simple, precum intregi sau stringuri, atunci comparatorul default este foarte convenabil. Insa pentru o clasa definita de utilizator, trebuie fie implementata interfata IComparable, fie creata o alta clasa care suporta IComparer si compara obiecte de tipul primei clase. A se vedea exemplul urmator, in care se utilizeaza ambele variante ale metodei Sort().

De notat ca o serie de colectii precum CollectionBase nu expun metode pentru sortare. Daca doriti sa sortati o colectie puternic tipizata care deriva din CollectionBase atunci trebuie sa implementati manual codul pentru sortare (sau sa salvati lista colectiei intr-un ArrayList si sa utilizati apoi metoda Sort() a colectiei ArrayList).

```
using System;      using System.Collections;          using System.Collections.Generic;
using System.Linq;             using System.Text;
namespace Sortare
{
      class Persoana : IComparable
      {
         public string Nume;     public int Varsta;

         public Persoana(string nume, int varsta)
         {  Nume = nume;     Varsta = varsta;   }

          public int CompareTo(object obiect)
          {
             if (obiect is Persoana)
             {
                Persoana altaPersoana = obiect as Persoana;
                return this.Varsta - altaPersoana.Varsta;
             }
             else
             { throw new ArgumentException("Objectul cu care se compara nu este o Persoana.");   }
          }
```

```
public class ComparaNumePersoane : IComparer
    {
        public static IComparer Default = new ComparaNumePersoane();

        public int Compare(object x, object y)
        {
            if (x is Persoana && y is Persoana)
            {
                return Comparer.Default.Compare(
                    ((Persoana)x).Nume, ((Persoana)y).Nume);
            }
            else
            {
                throw new ArgumentException("Cel putin unul din obiecte nu este Persoana.");
            }
        }
    }
```

```
class Program
    {


    static void Main(string[] args)
    {
       ArrayList lista = new ArrayList();
       lista.Add(new Persoana("Ilie", 30));
       lista.Add(new Persoana("Domnica", 25));
       lista.Add(new Persoana("Simion", 27));
       lista.Add(new Persoana("Safta", 22));

       Console.WriteLine("Persoane Nesortate:");
       for (int i = 0; i < lista.Count; i++)
       {
          Console.WriteLine("{0} ({1})",
             (lista[i] as Persoana).Nume, (lista[i] as Persoana).Varsta);
       }
       Console.WriteLine();
```

```
Console.WriteLine("Persoane sortate cu comparatorul default (dupa varsta):");
        lista.Sort();
        for (int i = 0; i < lista.Count; i++)
        {
            Console.WriteLine("{0} ({1})",
              (lista[i] as Persoana).Nume, (lista[i] as Persoana).Varsta);
        }
        Console.WriteLine();

        Console.WriteLine(
          "Persoane sortate cu comparatorul non-default (dupa nume):");
        lista.Sort(ComparaNumePersoane.Default);
        for (int i = 0; i < lista.Count; i++)
        {
            Console.WriteLine("{0} ({1})",
              (lista[i] as Persoana).Nume, (lista[i] as Persoana).Varsta);
        }

        Console.ReadKey();
    }
  }
}
```

Conversii. De fiecare data atunci cand a trebuit sa convertim un tip intr-un altul am utilizat un cast. Se poate insa proceda si altfel.

Asa cum este posibil sa supraincarcam operatorii matematici (a se vedea unul dintre cursurile anterioare), putem defini conversii implicite si explicite intre diverse tipuri. Aceste operatii sunt utile atunci cand dorim sa convertim un tip intr-un altul, fara a fi vreo legatura intre cele doua tipuri (spre exemplu, nu exista nici o relatie de mostenire sau nu implementeaza nici o interfata comuna).

Sa presupunem ca avem definita o conversie implicita a tipului ConvClass1 in ConvClass2. Aceasta inseamna ca putem scrie

```
ConvClass1 op1 = new ConvClass1();
ConvClass2 op2 = op1;
```

Daca avem o conversie explicita atunci:

```
ConvClass1 op1 = new ConvClass1();
ConvClass2 op2 = (ConvClass2)op1;
```

Ca exemplu a modului cum pot fi definiti operatorii de conversie sa consideram codul:

```
   public class ConvClass1
{
   public int val;
   public static implicit operator ConvClass2(ConvClass1 op1)
   {
            ConvClass2 valIntoarsa = new ConvClass2();
             valIntoarsa.val = op1.val;
            return valIntoarsa;
   }
}
public class ConvClass2
{
   public double val;
   public static explicit operator ConvClass1(ConvClass2 op1)
   {
            ConvClass1 valIntoarsa  = new ConvClass1();
            checked {valIntoarsa.val = (int)op1.val;};
            return valIntoarsa;
}
}
```

Aici ConvClass1 contine o variabile de tip int, iar ConvClass2 o variabila de tip double. Deoarece o valoare intreaga poate fi convertita implicit intr-una de tip double, definim un operator implicit de conversie a clasei ConvClass1 in ConvClass2. Pentru operatia inversa, definim un operator explicit. Cu acesti operatori de conversie, au sens urmatoarele secvente de cod:

```
ConvClass 1 op1=new ConvClass1();
op1.val=3;
ConvClass2 op2=op1;
```

si respectiv (utilizand conversia explicita):

```
ConvClass 2 op4=new ConvClass2();
op4.val=3e15;
ConvClass1 op3=(ConvClass1)op4;
```

Intrucat s-a utilizat cuvantul cheie checked in definitia conversiei explicite, vom obtine o exceptie in codul precedent intrucat valoarea parametrului op4 este prea mare pentru a fi memorata de o variabila de tip int.

Operatorul as. Acest operator este utilizat pentru a converti un tip intr-un tip referinta, utilizand sintaxa: <operand> as <type>

Aceasta operatie este posibila daca operand este de tip <type>, sau daca <operand> poate fi convertit implicit in <type> sau daca <operand> poate fi impachetat in tipul <type>. Daca nu este posibila nici o conversie atunci rezultatul este null.