IFRN

PROGRAMAÇÃO ORIENTADA A OBJETOS EM C#

Coleções de Objetos

Prof. Gilbert Azevedo

Objetivos

- Conhecer e utilizar coleções de objetos na programação de aplicações
  - Conceito de coleções
  - Grupos e tipos de coleções
  - Classes de coleções
  - Modelo de entidades com coleções

Coleções

- Grupos de objetos podem ser manipulados utilizando vetores e matrizes de objetos (Arrays) ou coleções de objetos
  - Vetores e matrizes de objetos se caracterizam pelo número fixo de objetos fortemente tipados
  - Coleções de objetos são flexíveis em relação ao número de elementos tendo seu tamanho ajustado dinamicamente
- Coleções são contêineres para objetos e são classificadas de acordo com sua funcionalidade
  - As classes de coleções definem operações e formas de acesso aos elementos
  - Utilizam Arrays e listas encadeadas na implementação

Grupos de Coleções

- Coleções comuns: System.Collections
  - Armazenam os elementos como objetos, utilizando o tipo object
  - Permitem que a coleção tenha objetos de classes diferentes
  - Necessitam constantemente de realização de type-cast
- Coleções genéricas: System.Collections.Generic
  - Utilizam parâmetros de tipo para definir o tipo dos elementos
  - São coleções type-safe e dispensa as operações de type-cast
- Coleções concorrentes: System.Collections.Concurrent
  - São coleções thread-safe

Tipos de Coleções

- Coleções são definidas de acordo com a sua funcionalidade
  - Métodos para realizar operações com os elementos
  - Propriedades e indexadores para acesso aos elementos
- Tipos de Coleções
  - Vetor de tamanho dinâmico
  - Fila: algoritmo FIFO – first in, first out
  - Pilha: algoritmo LIFO – last in, first out
  - Dicionário não ordenado
  - Dicionário ordenado

Classes de Coleções

- Classes de Collections – usam o tipo object para referenciar os elementos
- Classes de Collections.Generic – usam parâmetros de tipo para definir o tipo dos elementos

Coleção	Collections	Collections.Generic
Vetor dinâmico	ArrayList	List
Fila	Queue	Queue
Pilha	Stack	Stack
Dic. não ordenado	Hashtable	Dictionary
Dicionário ordenado	SortedList	SortedList, SortedDictionary

Vetores Dinâmicos

- Classes ArrayList e List<T>
- Implementam um vetor de tamanho dinâmico, cujos elementos são acessados através de um índice
- Principais Métodos
  - Add, Insert – Adiciona um elemento no vetor (final, posição dada)
  - Clear – Remove todos os elementos do vetor
  - Remove, RemoveAt – Remove um elemento (primeira ocorrência, posição dada)
  - Contains – Verifica se um elemento está presente no vetor (lógico)
  - IndexOf – Retorna o índice de um elemento (inteiro)
  - CopyTo – Copia elementos do vetor para um Array
  - Sort – Ordena os elementos do vetor

Vetores Dinâmicos

- Principais Propriedades
  - Count – Informa a quantidade de elementos
  - Indexador [ ] – Indexa os elementos utilizando um índice inteiro
- Algumas Interfaces Implementadas
  - ICollection: Count, CopyTo
  - IEnumerable: foreach
  - IList: [ ], Add, Clear, Contains, IndexOf, Insert, Remove, RemoveAt

Array e ArrayList

- Array de objetos (inteiros e strings)
  - Tamanho fixo, [ ], Length, [ ], foreach

```
public static void Main () {
  object[] v = new object[5];
  v[0] = "Coleções";
  v[1] = "em";
  v[2] = "C#";
  v[3] = 8;
  v[4] = 2020;
  Console.WriteLine(v.Length);
  Console.WriteLine(v[0]);
  foreach (object x in v)
    Console.WriteLine(x);
}
```

- ArrayList de objetos
  - Tam. dinâmico, Add, Count, [ ], foreach

```
public static void Main () {
  ArrayList w = new ArrayList();
  w.Add("Coleções");
  w.Add("em");
  w.Add("C#");
  w.Add(8);
  w.Add(2020);
  Console.WriteLine(w.Count);
  Console.WriteLine(w[0]);
  foreach (object x in w)
    Console.WriteLine(x);
}
```

ArrayList e List<T>

- ArrayList de objetos (inteiros)
  - Box/unbox, type-cast

```
public static void Main () {
  ArrayList v = new ArrayList();
  v.Add(2);
  v.Add(4);
  v.Add(6);
  v.Add(8);
  v.Add(10);
  int sv = 0;
  for(int i = 0; i < v.Count; i++)
    sv = sv + (int)v[i];
  Console.WriteLine(sv);
}
```

- List<T> de inteiros
  - Sem box/unbox, sem type-cast

```
public static void Main () {
  List<int> w = new List<int>();
  w.Add(2);
  w.Add(4);
  w.Add(6);
  w.Add(8);
  w.Add(10);
  int sw = 0;
  for(int i = 0; i < w.Count; i++)
    sw = sw + w[i];
  Console.WriteLine(sw);
}
```

Filas

- Queue e Queue<T>
- Implementa uma fila (FIFO) de objetos onde o primeiro elemento a entrar é o primeiro a sair
- Principais Métodos e Propriedades
  - Clear – Remove todos os elementos da fila
  - Dequeue – Remove e retorna o elemento no início da fila
  - Enqueue – Insere um elemento no final da fila
  - Peek – Retorna o elemento no início da fila sem removê-lo
  - Count – Propriedade que informa a quantidade de elementos na fila
- Interfaces Implementadas
  - ICollection, IEnumerable

Queue e Queue<T>

- Fila de inteiros e strings
  - Box/unbox

```
public static void Main () {
  Queue f1 = new Queue();
  f1.Enqueue("Coleções");
  f1.Enqueue("em");
  f1.Enqueue("C#");
  f1.Enqueue(8);
  f1.Enqueue(2020);
  while (f1.Count > 0)
    Console.WriteLine(f1.Dequeue());
}
```

- Fila genérica de strings
  - Elementos do mesmo tipo

```
public static void Main () {
  Queue<string> f2 = new
    Queue<string>();
  f2.Enqueue("Coleções");
  f2.Enqueue("em");
  f2.Enqueue("C#");
  f2.Enqueue("8");
  f2.Enqueue("2020");
  while (f2.Count > 0)
    Console.WriteLine(f2.Dequeue());
}
```

Pilhas

- Stack e Stack<T>
- Implementa uma pilha (LIFO) de objetos onde o último elemento a entrar é o primeiro a sair
- Principais Métodos e Propriedades
  - Clear – Remove todos os elementos da pilha
  - Peek – Retorna o objeto no início da pilha sem removê-lo
  - Pop – Remove e retorna o elemento no início da pilha
  - Push – Insere um elemento no início da pilha
  - Count – Propriedade que informa a quantidade de elementos na fila
- Interfaces Implementadas
  - ICollection, IEnumerable

Stack e Stack<T>

- Pilha de inteiros e strings
  - Box/unbox

```
public static void Main () {
  Stack p1 = new Stack();
  p1.Push("Coleções");
  p1.Push("em");
  p1.Push("C#");
  p1.Push(8);
  p1.Push(2020);
  while (p1.Count > 0)
    Console.WriteLine(p1.Pop());
}
```

- Pilha genérica de strings
  - Elementos do mesmo tipo

```
public static void Main () {
  Stack<string> p2 = new
    Stack<string>();
  p2.Push("Coleções");
  p2.Push("em");
  p2.Push("C#");
  p2.Push("8");
  p2.Push("2020");
  while (p2.Count > 0)
    Console.WriteLine(p2.Pop());
}
```

Dicionários não Ordenados

- Hashtable e Dictionary<TKey, TValue>
- Implementa um dicionário não ordenado onde os elementos são acessados utilizando uma chave
- Cada item do dicionário é um par chave – valor
  - Na Hashtable, a chave e o valor do item são objetcs: DictionaryEntry
  - No Dictionary, a chave é do tipo TKey e o valor é TValue: KeyValuePair
  - TKey e TValue são parâmetros de tipo informados ao instanciar o dicionário
- Exceções
  - A chave de um elemento do dicionário não pode ser nula (null) nem duplicada
- Interfaces
  - ICollection, IDictionary, IEnumerable

Dicionários não Ordenados

- Principais Métodos
  - Add – Adiciona um item (par chave-valor)
  - Clear – Remove todos os itens
  - Remove – Remove um item dada a chave
  - Contains, ContainsKey – Verifica se uma chave existe
  - ContainsValue – Verifica se um valor existe
- Principais Propriedades
  - Count – Informa a quantidade de elementos
  - Indexador [ ] – Indexa os itens utilizando o tipo da chave, possibilita incluir e alterar o valor de uma chave
  - Keys – Coleção com as chaves do dicionário
  - Values – Coleção com os valores do dicionário

Hashtable e Dictionary

- Dicionário
  - Chave/elemento - object

```
Hashtable h = new Hashtable();
h["Brasil"] = 5;
h["Itália"] = 4;
h["Alemanha"] = 4;
h["Argentina"] = 2;
h.Add("Uruguai", 2);
h.Add("França", 2);
h.Add("Inglaterra", 1);
h.Add("Espanha", 1);
foreach (DictionaryEntry x in h)
  Console.WriteLine($"{x.Key} =
    {x.Value} Título(s)");
```

- Dicionário genérico
  - Chave string, elemento inteiro

```
Dictionary<string, int> d = new
  Dictionary<string, int>();
d["Brasil"] = 5;
d["Itália"] = 4;
d["Alemanha"] = 4;
d["Argentina"] = 2;
d.Add("Uruguai", 2);
d.Add("França", 2);
d.Add("Inglaterra", 1);
d.Add("Espanha", 1);
foreach (KeyValuePair<string, int> x in d)
  Console.WriteLine($"{x.Key} = {x.Value}
    Título(s)");
```

Dicionários Ordenados

- SortedList, SortedList<TKey, TValue>, SortedDictionary<TKey, TValue>
- Implementa um dicionário ordenado onde os elementos são acessados utilizando uma chave
- SortedList utiliza arrays e SortedDictionary utiliza árvore binária
- Funcionalidade semelhante ao dicionário não ordenado, mas mantém as chaves ordenadas
- Devido a ordenação, as chaves dos elementos devem implementar a interface *IComparable*

Hashtable e SortedList

- Dicionário não ordenado
  - Chave/elemento - object

```
Hashtable h = new Hashtable();
h["Brasil"] = 5;
h["Itália"] = 4;
h["Alemanha"] = 4;
h["Argentina"] = 2;
h.Add("Uruguai", 2);
h.Add("França", 2);
h.Add("Inglaterra", 1);
h.Add("Espanha", 1);
foreach (DictionaryEntry x in h)
  Console.WriteLine($"{x.Key} =
    {x.Value} Título(s)");
```

- Dicionário ordenado
  - Chave/elemento - object

```
SortedList s = new SortedList();
s["Brasil"] = 5;
s["Itália"] = 4;
s["Alemanha"] = 4;
s["Argentina"] = 2;
s.Add("Uruguai", 2);
s.Add("França", 2);
s.Add("Inglaterra", 1);
s.Add("Espanha", 1);
foreach (DictionaryEntry x in s)
  Console.WriteLine($"{x.Key} = {x.Value}
    Título(s)");
```

Hashtable e SortedList

- Dicionário não ordenado
  - Chave/elemento - object

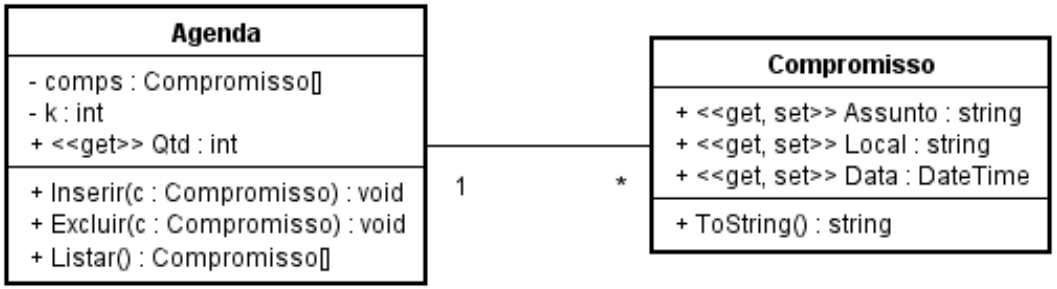
```
França = 2 Título(s)
Brasil = 5 Título(s)
Espanha = 1 Título(s)
Itália = 4 Título(s)
Inglaterra = 1 Título(s)
Argentina = 2 Título(s)
Alemanha = 4 Título(s)
Uruguai = 2 Título(s)
```

- Dicionário ordenado
  - Chave/elemento - object

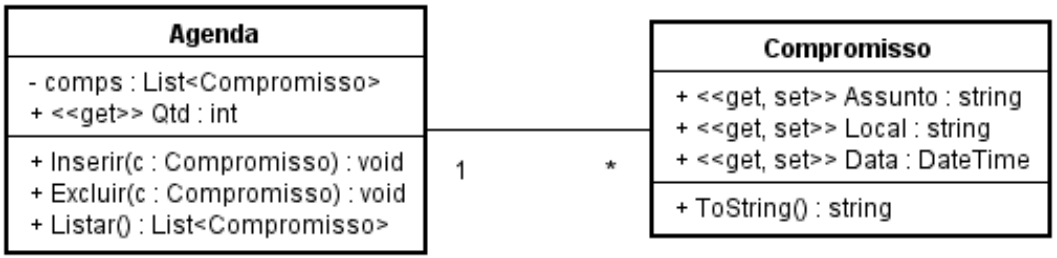
```
Alemanha = 4 Título(s)
Argentina = 2 Título(s)
Brasil = 5 Título(s)
Espanha = 1 Título(s)
França = 2 Título(s)
Inglaterra = 1 Título(s)
Itália = 4 Título(s)
Uruguai = 2 Título(s)
```

Modelando Entidades com Coleções

- Vetores dinâmicos e genéricos são muito utilizados no modelo de entidades



- Arrays são substituídos por vetores dinâmicos: List<T>



Modelando Entidades com Coleções

- O vetor dinâmico dispensa o controle de nº de elementos inseridos

```
class Agenda {
  private List<Compromisso> comps = new List<Compromisso>();
  public int Qtd { get => comps.Count; }
  public void Inserir(Compromisso c) {
    comps.Add(c);
  }
  public void Excluir(Compromisso c) {
    comps.Remove(c);
  }
  public List<Compromisso> Listar() {
    return comps;
  }
}
```

Referências

- Microsoft Visual C# 2010 – Passo a passo, John Sharp, Bookman, 2010
- Guia de Programação em C# – Coleções
  - https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/concepts/collections