FICHA 6
COLEÇÕES

MODALIDADE:	Aprendizagem +	Escolha um item.	
CURSO:	Tecnologias de Programação de Sistemas de Informação		
UC:	Desenvolver programas complexos em linguagem de programação estruturada	CÓDIGO UC:	UC00607
FORMADOR/A:	Bruno Silva	DATA:	

OBJETIVOS

- Perceber a implementação das coleções com a linguagem C#

Introdução as coleções

Arrays (como já falado anteriormente nos casos dos vetores e matrizes) e **Coleções** são objetos que atuam como contentores de informação.

A utilização de arrays como contentores de dados é já nossa conhecida e utilizada extensivamente (lembrem-se a utilização de arrays para guardar notas das disciplinas, listas de números, etc.).

Os dados guardados nas coleções podem ser depois processados (cálculos sobre esses dados, decisões tomadas com base nos conteúdos dos dados, etc.)

As coleções são ***mais flexíveis que os vetores e as matrizes***, e são ideais para trabalhar com um ***número indeterminado de elementos***.

O ***nome coleções*** deriva do facto de se tratar de uma ***interface de programação*** para tratamento de ***coleções de objetos***.

Tal como uma matriz, uma ***coleção permite guardar múltiplos elementos***, e, ao ***contrário*** das matrizes, ***as coleções não têm um tamanho fixo***.

O ***tamanho de uma coleção aumenta automaticamente*** quando são adicionados elementos.

Similares em:

- Ambos permitem guardar múltiplos elementos.

Diferentes em:

- As matrizes estão embutidas na linguagem C#. As coleções são classes fornecidas pelo Visual Studio.
- As coleções oferecem métodos para efetuar operações (adicionar, remover, etc.).
- As matrizes têm tamanho fixo. As coleções têm tamanho variável.

Principais coleções da Plataforma .NET

Classe Tracional ***(Collections)***	***Classe Genérica*** ***(Collections.Generic)***	***Descrição***
ArrayList	***List<T>***	Funciona como uma tabela cujo tamanho cresce automaticamente.
---	***LinkedList<T>***	Armazena os objetos linearmente, utilizando uma lista ligada (ligando o valor antigo com o próximo valor)
HashTable	***Dictionary<Key, Value>***	Representa uma coleção de pares (chave, valor). A sua organização é baseada na chave, com acesso muito eficiente. Sem relação de ordem na chave.
SortedList	***SortedList<Key,Value>***	Semelhante a Dictionary, mas os pares (chave, valor) são mantidos ordenados, baseados no valor da chave.
Queue (Fila)	***Queue<T>***	Representa uma coleção de objetos em que o primeiro a ser colocado na coleção é o primeiro a sair ***(FIFO) – First In First Out***.
Stack (Pilha)	***Stack<T>***	Representa uma coleção de objetos em que o último a ser colocado na coleção é o primeiro a sair ***(LIFO) – Last In First Out***.

Exercícios

1 – Crie **uma fila de espera** (Queue) de **conteúdos textuais** (string), para armazenar nomes. De seguida, deve fazer os seguintes passos:

- **Introduzir** 3 nomes com a função **Enqueue()**;
- **Mostrar** quantos elementos existem na fila com a propriedade **Count**;
- **Mostrar** qual **é o primeiro elemento da fila** com a função **Peek()**;
- Verifique se a lista tem um nome com a função **Contains()**;
- **Percorra a fila** com uma **estrutura de repetição while()** e processe cada elemento com a função **Dequeue()**;

2 – Crie **uma pilha de dados** (Stack) de **números inteiros** (int), para armazenar valores numéricos inteiros. De seguida, deve fazer os seguintes passos:

- **Introduzir** 3 números inteiros com a função **Push()**;
- **Mostrar** quantos elementos existem na fila com a propriedade **Count**;
- **Mostrar** qual **é o primeiro elemento da fila** com a função **Peek()**;
- **Verifique** se a lista **contém** o número 5 com a função **Contains()**;
- **Percorra a pilha** com uma **estrutura de repetição while()** e processe cada elemento com a função **Pop()**

3 – Crie **uma lista** (List) de **números inteiros** (int) com o nome **lista_inteiros**, para armazenar valores numéricos inteiros. De seguida, deve fazer os seguintes passos:

- **Introduzir** 3 números inteiros com a função **Add()**;
- **Mostrar** quantos elementos existem na fila com a propriedade **Count**;
- **Verifique** se a lista **contém** o número 7 com a função **Contains()**;
- **Percorra a lista** com uma **estrutura de repetição foreach()** e processe cada elemento individualmente.

  Na construção do **foreach**(), temos de indicar a seguinte e **para cada valor que percorrer na coleção** (**lista_inteiros**):
  - tem de **armazenar o elemento numa variável temporária** (como temos uma lista de inteiros, indicamos **int valor** (onde **valor** é um nome sugestivo para a variável));
  - indicar que essa informação está dentro (**in**) da coleção que está a percorrer (neste exemplo é a coleção **lista_inteiros**):

```
//Percorrer a lista com a estrutura de repetição FOREACH
foreach (int valor in lista_inteiros)
{
    Console.WriteLine(valor);
}
```

- **Ordene** a lista de forma crescente com a função **Sort();**
- **Remova** (se possível) o número 7 da lista com a função **Remove();**
- **Percorra novamente a lista** com uma **estrutura de repetição foreach()** e processe cada elemento individualmente;

4 – Crie **uma lista** (List) baseado na classe **Produto** (feito na ficha 5) com o nome **lista_produtos**, para armazenar produtos. De seguida, deve fazer os seguintes passos:

- **Introduzir** 3 produtos com a função **Add()**;
- **Mostrar** quantos elementos existem na fila com a propriedade **Count**;
- **Verifique** se a lista **contém** o código ID igual a 3 com a função **Any()**;
- **Percorra a lista** com uma **estrutura de repetição foreach()** e processe cada elemento individualmente com base na classe Produto;
  Na construção do **foreach**(), temos de indicar a seguinte e **para cada valor que percorrer na coleção** (**lista_inteiros**):
  - tem de **armazenar o elemento numa variável temporária** (neste caso como estamos a trabalhar com uma classe (Produto), colocamos a instrução **Produto item** (onde **item** é um nome sugestivo para a variável));
  - indicar que essa informação está dentro (**in**) da coleção que está a percorrer (neste exemplo é a coleção **lista_produtos**);
  - Utilizar o método que desenvolvemos **Mostra_Produto()**, para mostrar as informações;

```
//Percorrer a lista com a estrutura de repetição FOREACH
foreach (Produto item in lista_produtos)
{
    Console.WriteLine(item.Mostra_Produto());
}
```

- **Remova** (se possível) o produto com o código ID igual a 3 da lista com a função **RemoveAll();**
- **Percorra novamente a lista** com uma **estrutura de repetição foreach()** e processe cada elemento individualmente;

5 – Crie **uma lista ordenada** (SortedList) com a **chave** baseada num número inteiro e o **valor** baseado na classe **Livro** (feito na ficha 5) com o nome **lista_livros**, para armazenar livros. De seguida, deve fazer os seguintes passos:

- **Introduzir** 3 livros com a função **Add()**;
- **Mostrar** quantos elementos existem na fila com a propriedade **Count**;
- **Verifique** se a lista **contém** o código 2 com a função **ContainsKey()**;
- **Remova** (se possível) o livro com a **chave** 2 da lista com a função **Remove();**
- **Percorra novamente a lista** com uma **estrutura de repetição foreach()** e processe cada elemento individualmente (mostrar qual a chave e os valores do livro);