

# Apresentação

---

Curso:

Programação .NET I

Aula 04:

Objetivos:

Arrays primitivos;

Instrução *foreach*

Tratamento de erros

Instruções verificadas;

# Coleções de dados (*Arrays*)

---

Já vimos a declaração e utilização de dados primitivos no C#. Agora vamos iniciar a nossa jornada para utilização de uma coleção de dados, ou seja, guardar mais de um valor no mesmo objeto-referência (array).

Imagine que precisamos guardar ao invés de um único valor, n-valores... Para isso, são utilizadas as coleções de dados. Vamos ver como declarar um array de tipos primitivos no C#.

# Coleções de dados (*Arrays*)

---

Declarando e inicializando um array de uma dimensão:

```
tipo[] nomeArray;  
nomeArray = new tipo[TAMANHO];
```



# Coleções de dados (*Arrays*)

Inicializando um array de int de uma dimensão:

Inicialização com valores padrões:

```
int[] notas;  
notas = new int[];
```

Inicialização com valores pré-fixados:

```
int[] valores = {10, 3, 5, 3};
```

# Coleções de dados (*Arrays*)

Acessando um array de int de uma dimensão:

```
int valor = 0;  
int[] valores = {10, 3, 42, 3};
```

```
valor = valores[2]; //valor recebeu o número “42”
```

# Coleções de dados (*Arrays*)

---

Modificando um elemento de um array de int de uma dimensão:

```
int[] valores = {10, 3, 42, 3};
```

```
valores[3] = 17
```

//a posição 3 recebeu o número “17”. valores ficou da seguinte forma:

```
{10, 3, 42, 17}
```

# Coleções de dados (*Arrays*)

---

Declarando e inicializando um array de duas dimensões:

```
tipo[,] nomeMatriz;  
nomeMatriz = new tipo[TAM1,TAM2];
```

# Coleções de dados (*Arrays*)

Inicializando um array de int de duas dimensões:

Inicialização com valores padrões:

```
int[,] matrizA;  
matrizA = new tipo[4,2];
```

Inicialização com valores pré-fixados:

```
int[,] matrizA = new int[4,2]{{0,1}, {2,3}, {4,5}, {6,7}};  
int[,] matrizA = {{0,1}, {2,3}, {4,5}, {6,7}};
```



# Coleções de dados (*Arrays*)

---

Inicialização com valores pré-fixados (mais elegante!):

```
int[, ] matrizA = new int[4,2]
{
    {0,1},
    {2,3},
    {4,5},
    {6,7}
};
```

```
int[, ] matrizA =
{
    {0,1},
    {2,3},
    {4,5},
    {6,7}
};
```

# Coleções de dados (*Arrays*)

---

Acessando um array de int de duas dimensões:

```
int valor = 0;  
int[,] matrizA =  
{  
    {0,1},  
    {2,3},  
    {4,5},  
    {6,7}  
};
```

```
valor = matrizA[2,0]; //valor recebeu o número “4”
```

# Coleções de dados (*Arrays*)

---

Modificando um array de int de duas dimensões:

```
int[, ] matrizA =  
{  
    {0,1},  
    {2,3},  
    {4,5},  
    {6,7}  
};
```

```
matrizA[2,0] = 17
```



# Coleções de dados (*Arrays*)

---

Resultado da modificação de um array de int de duas dimensões:

```
int[, ] matrizA =  
{  
    {0,1},  
    {2,3},  
    {17,5},  
    {6,7}  
};
```

matrizA[2,0] = 17 //a linha 2 e coluna 0 recebeu o valor inteiro “17”

# Coleções de dados (*Arrays*)

---

Apontamentos de um Array:

- São tipos-referência (*heap*);
- Valores padrões (0 (zero), null, false);
- O tamanho de uma instância não precisa ser uma constante, pode ser calculado em tempo de execução;
- O intervalo de índice inicia em 0 (zero) e vai até TAMANHO-1;
- O método *objArray.GetUpperBound(DIM)*; Retorna o valor máximo da dimensão específica (Linha 0, Coluna 1);

# Exercício Prático (array)

---

Exercício 1a:

Name: dia\_semana\_array

Solution name: Nome\_Sobrenome-Aula04

Crie um programa que leia um inteiro, imprima o nome do dia da semana correspondente e guarde o nome retornado num array. O programa deve ler 5 vezes, ao final, deve imprimir\*\* todos os nomes de dias das semanas selecionados e o respectivo índice.

1 = Domingo ... 7 = Sábado

\* Utilize uma array para guardar todos os 5 nomes selecionados.

\*\* A impressão do array deve ser feita utilizando uma das três estruturas apresentadas anteriormente (while, for, do-while).

# Resposta - Exercício Prático (array)

```
string[] nomesDiaSemana = new string[5];
string nomeDiaSemana = "";
int diaSemana = 0;

for (int i = 0; i < 5; i++)
{
    Console.WriteLine("Programa de estrutura while - Leitura: " + i);
    Console.WriteLine("Digite um numero inteiro entre 1 e 7");
    diaSemana = Int32.Parse(Console.ReadLine());

    if (diaSemana == 1)
        nomeDiaSemana = "Domingo";
    else if (diaSemana == 2)
        nomeDiaSemana = "Segunda-feira";
    else if (diaSemana == 3)
        nomeDiaSemana = "Terça-feira";
    else if (diaSemana == 4)
        nomeDiaSemana = "Quarta-feira";
    else if (diaSemana == 5)
        nomeDiaSemana = "Quinta-feira";
    else if (diaSemana == 6)
        nomeDiaSemana = "Sexta-feira";
    else if (diaSemana == 7)
        nomeDiaSemana = "Sábado";
    else
    {
        nomeDiaSemana = "Desconhecido!";
    }

    Console.WriteLine("Dia escolhido: " + nomeDiaSemana);
    nomesDiaSemana[i] = nomeDiaSemana;
}

for (int i = 0; i < 5; i++)
{
    Console.WriteLine("Dia escolhido [" + i + "]: " + nomesDiaSemana[i]);
}

Console.ReadKey(true);
```



# Exercício Prático (array)

---

Exercício 1b:

Name: dia\_semana\_array\_copy

Solution name: Add to solution

Com base no programa anterior, copie todo o conteúdo do array impresso ao final para uma nova instância de array.

Imprima\*, ao final do programa, o conteúdo do array original e do array de cópia e seus respectivos índices.

\* A impressão do array deve ser feita utilizando uma das três estruturas apresentadas anteriormente (while, for, do-while).



# Resposta - Exercício Prático (array)

```
string[] nomesDiaSemana = new string[5];
string nomeDiaSemana = "";
int diaSemana = 0;

for (int i = 0; i < 5; i++)
{
    Console.WriteLine("Programa de estrutura while - Leitura: " + i);
    Console.WriteLine("Digite um numero inteiro entre 1 e 7");
    diaSemana = Int32.Parse(Console.ReadLine());

    if (diaSemana == 1)
        nomeDiaSemana = "Domingo";
    else if (diaSemana == 2)
        nomeDiaSemana = "Segunda-feira";
    else if (diaSemana == 3)
        nomeDiaSemana = "Terça-feira";
    else if (diaSemana == 4)
        nomeDiaSemana = "Quarta-feira";
    else if (diaSemana == 5)
        nomeDiaSemana = "Quinta-feira";
    else if (diaSemana == 6)
        nomeDiaSemana = "Sexta-feira";
    else if (diaSemana == 7)
        nomeDiaSemana = "Sábado";
    else
    {
        nomeDiaSemana = "Desconhecido!";
    }

    Console.WriteLine("Dia escolhido: " + nomeDiaSemana);
    nomesDiaSemana[i] = nomeDiaSemana;
}

string[] copiaNomesDiaSemana = new string[nomesDiaSemana.Length];
nomesDiaSemana.CopyTo(copiaNomesDiaSemana, 0);

for (int i = 0; i < 5; i++)
{
    Console.WriteLine("Dia escolhido[" + i + "]: " + nomesDiaSemana[i] +
        "\tCopia[" + i + "]: " + copiaNomesDiaSemana[i]);
}
Console.ReadKey(true);
```

# Estruturas de repetição (*foreach*)

Utilizando as estruturas de repetição anteriores, notou-se uma necessidade de utilizar uma variável auxiliar para ser utilizada como controle da expressão lógica.

Com a implementação da tecnologia de iteradores (Visual Studio 2005 C# 2.0), criou-se a instrução *foreach*.

# Estruturas de repetição (*for-each*)

## Escopo da instrução *foreach*:

```
foreach (tipo varElem in objArray)
{
    BLOCO DE INSTRUÇÕES
}
```

Ou:

```
foreach (tipo varElem in objArray)
    LINHA DE INSTRUÇÃO;
```

# Estruturas de repetição (*for-each*)

## Apontamentos para utilização do *for-each*:

- A *foreach* itera por todo o array. Se você quiser iterar em partes do array, a instrução *for* é a melhor opção;
- Sempre começa do índice 0 (zero) até o índice Tamanho - 1). Para uma iteração reversa é melhor utilizar o *for*;
- Se é necessário saber o índice do elemento, melhor utilizar o *for*;
- Se precisa modificar elementos do array, utilize a instrução *for*. O *for-each* possui uma cópia somente-leitura de cada elemento do array

# Exercício Prático (*for-each*)

---

Exercício 2a:

Name: dia\_semana\_foreach

Solution name: Add to solution

Com base no programa anterior, altere-o para utilizar a instrução *for-each* para imprimir ao final do programa o conteúdo do array original e do array de cópia.

# Resposta - Exercício Prático (array)

```
string[] nomesDiaSemana = new string[5];
string nomeDiaSemana = "";
int diaSemana = 0;

for (int i = 0; i < 5; i++)
{
    Console.WriteLine("Programa de estrutura while - Leitura: " + i);
    Console.WriteLine("Digite um numero inteiro entre 1 e 7");
    diaSemana = Int32.Parse(Console.ReadLine());

    if (diaSemana == 1)
        nomeDiaSemana = "Domingo";
    else if (diaSemana == 2)
        nomeDiaSemana = "Segunda-feira";
    else if (diaSemana == 3)
        nomeDiaSemana = "Terça-feira";
    else if (diaSemana == 4)
        nomeDiaSemana = "Quarta-feira";
    else if (diaSemana == 5)
        nomeDiaSemana = "Quinta-feira";
    else if (diaSemana == 6)
        nomeDiaSemana = "Sexta-feira";
    else if (diaSemana == 7)
        nomeDiaSemana = "Sábado";
    else
    {
        nomeDiaSemana = "Desconhecido!";
    }

    Console.WriteLine("Dia escolhido: " + nomeDiaSemana);
    nomesDiaSemana[i] = nomeDiaSemana;
}

string[] copiaNomesDiaSemana = new string[nomesDiaSemana.Length];
nomesDiaSemana.CopyTo(copiaNomesDiaSemana, 0);

foreach (string nome in nomesDiaSemana)
    Console.WriteLine("Dia escolhido: " + nome);
foreach (string nome in nomesDiaSemana)
    Console.WriteLine("Dia escolhido - copia: " + nome);
```

# Exercício Prático (array)

---

Exercício 2b:

Name: nome\_array

Solution name: Add to solution

Com base no conhecimento em *array*, faça um programa que leia 5 nomes, porém:

- Dê mensagem de erro caso a primeira letra seja igual a primeira letra do SEU nome (maiúscula ou minúscula);
- Guarde todos os nomes digitados (caso não entrem na primeira hipótese) e imprima-os ao final do programa.

# Resposta - Exercício Prático (array)

---

```
const int totalNomes = 3;

string[] nomes = new string[totalNomes];
string nome;

for (int i = 0; i < totalNomes; i++)
{
    Console.WriteLine("Programa de array - Leitura: " + i);
    Console.WriteLine("Digite um nome: ");
    nome = Console.ReadLine();

    if ((nome[0] == 'A') || (nome[0] == 'a')) //if (nome.ToUpper()[0] == 'A')
    {
        i--;
        Console.WriteLine("O nome não pode iniciar com a letra A");
    }
    else
    {
        nomes[i] = nome;
    }
}

foreach (string n in nomes)
    Console.WriteLine("Nome gravado: " + n);

Console.ReadKey(true);
```



# Gerenciamento de erros (*try-catch*)

---

Erros podem acontecer em qualquer tarefa, mas e quando se trata de algo que “Não” pode dar errado? O caminho é aprender a gerenciar os erros e seguir em frente.

Diferente do que era utilizado antigamente, no que tange o mapeamento de erros, em que se tinha uma variável global que retornava um valor e o programador criava uma lógica paralela para fazer o tratamento, hoje, temos um novo tipo de abordagem.

# Gerenciamento de erros (*try-catch*)

---

No C# e na maioria das linguagens orientadas a objeto modernas, temos a implementação de “Exceptions”.

- São classes;
  - Possuem construtores, métodos, propriedades e eventos;
- São filhas da classe *System.Exception*;

# Gerenciamento de erros (*try-catch*)

---

Para gerenciarmos os erros ocorridos num programa C#, utilizamos do bloco “*try-catch*”. Este bloco consiste em rodar um trecho de código que será “supervisionado”, quando algum erro ocorre, a *Runtime* dispara uma Exceção.

Quando uma Exceção (Exception) é disparada (chamada/instanciada) pela Runtime, caso o código esteja escrito dentro de um bloco “*try-catch*”, a mesma irá procurar por um “*catch*” que trate a determinada Exceção disparada.

# Gerenciamento de erros (try-catch)

Escopo do bloco “try-catch” (sem instância da exceção):

```
try
{
    //BLOCO DE CÓDIGO SUPERVISIONADO
}
catch (ClasseExcecao)
{
    //BLOCO DE CÓDIGO PARA TRATAMENTO DA EXCEÇÃO
}
```

# Gerenciamento de erros (try-catch)

Escopo do bloco “try-catch”:

```
try
{
    //BLOCO DE CÓDIGO SUPERVISIONADO
}
catch (ClasseExcecao objExcecao)
{
    //BLOCO DE CÓDIGO PARA TRATAMENTO DA EXCEÇÃO
}
```

# Exercício Prático (try-catch)

---

Exercício 3a:

Name: pessoa\_try-catch

Solution name: Add to solution

Com base no exercício anterior “nome\_array” (mantendo as mesmas características). Acrescente a leitura de idade.

O programa deve tratar erros de preenchimento caso digitem letras no campo idade.

O programa deve imprimir os nomes e idades respectivas ao final do programa.

# Resposta - Exercício Prático (try-catch)

```
const int totalNomes = 3;

string[] nomes = new string[totalNomes];
int[] idades = new int[totalNomes];
string nome;
int idade;

for (int i = 0; i < totalNomes; i++)
{
    try
    {
        Console.WriteLine("Programa de array - Leitura: " + i);
        Console.Write("Digite um nome: ");
        nome = Console.ReadLine();

        if (nome.ToUpper()[0] == 'A')
        {
            i--;
            Console.WriteLine("O nome não pode iniciar com a letra A");
        }
        else
        {
            Console.Write("Digite uma idade: ");
            idade = Int32.Parse(Console.ReadLine());

            nomes[i] = nome;
            idades[i] = idade;
        }
    }
    catch (FormatException)
    {
        i--;
        Console.WriteLine("Digite um valor válido!");
    }
}

for (int i = 0; i < totalNomes; i++)
{
    Console.Write("Nome: " + nomes[i]);
    Console.WriteLine("\tIdade: " + idades[i]);
}

Console.ReadKey(true);
}
```



# Gerenciamento de erros (*try-catch*)

---

Múltiplos manipuladores “*catch*”:

```
try
{
    //BLOCO DE CÓDIGO SUPERVISIONADO
}
catch (ClasseExcecao1 objExcecao)
{
    //BLOCO DE CÓDIGO PARA TRATAMENTO DA EXCEÇÃO TIPO 1
}
catch (ClasseExcecao2 objExcecao)
{
    //BLOCO DE CÓDIGO PARA TRATAMENTO DA EXCEÇÃO TIPO 2
}
```



# Exercício Prático (try-catch)

---

Exercício 3b:

Name: `pessoa_try-catch_multiplo`

Solution name: Add to solution

Com base no exercício anterior. Acrescente um catch para tratar a análise de primeira letra quando a variável nome está vazia (“”).

O programa deve imprimir os nomes e idades respectivas ao final do programa.

# Resposta - Exercício Prático (try-catch)

---

```
...  
  
try  
{  
    Console.WriteLine("Programa de array - Leitura: " + i);  
    Console.Write("Digite um nome: ");  
    nome = Console.ReadLine();  
  
    if (nome.ToUpper()[0] == 'A')  
    {  
        i--;  
        Console.WriteLine("O nome não pode iniciar com a letra A");  
    }  
    else  
    {  
        Console.Write("Digite uma idade: ");  
        idade = Int32.Parse(Console.ReadLine());  
  
        nomes[i] = nome;  
        idades[i] = idade;  
    }  
}  
catch (FormatException)  
{  
    i--;  
    Console.WriteLine("Digite um valor válido!");  
}  
catch (IndexOutOfRangeException)  
{  
    i--;  
    Console.WriteLine("O nome não pode ser vazio!");  
}  
}
```

# Gerenciamento de erros (*try-catch*)

---

Como sabemos, as Classes de exceções são filhas da Classe `System.Exception` (é tipo uma `Object` das exceções).

Quando queremos pegar qualquer erro que possa ocasionalmente acontecer (e não é esperado), podemos utilizar um `catch` para o nível mais abstrato (porém sem detalhes) e manipular o erro.

# Gerenciamento de erros (*try-catch*)

---

Escopo do bloco “try-catch” para Exception (mais abstrato, porém menos detalhado):

```
catch (Exception objEx)
{
    //BLOCO DE CÓDIGO PARA TRATAMENTO DA EXCEÇÃO
}
```

Ou:

```
catch
{
    //BLOCO DE CÓDIGO PARA TRATAMENTO DA EXCEÇÃO
}
```

# Gerenciamento de erros (*try-catch*)

---

E se implementarmos um try-catch desta maneira?

```
try
{
    //BLOCO DE CÓDIGO SUPERVISIONADO
}
catch (Exception ex)
{
    Console.Write("Erro desconhecido!");
}
catch (IndexOutOfRangeException ex)
{
    Console.WriteLine(ex.ToString());
}
```



# Gerenciamento de erros (*try-catch*)

---

O Visual Studio não deixa compilar, pois verifica que um manipulador *catch* anterior já pega qualquer exceção que vier (está num nível mais abstrato).

O correto a se fazer, é inverter a ordem do *catch* mais especialista (menos abstrato) e coloca-lo antes.

PS: A Runtime verifica sequencialmente os blocos de *catch*, ou seja, o primeiro que “casar” com a exceção, é executado.

# Gerenciamento de erros (*try-catch*)

---

O mais correto a ser feito:

```
try
{
    //BLOCO DE CÓDIGO SUPERVISIONADO
}
catch (IndexOutOfRangeException ex)
{
    Console.WriteLine(ex.ToString());
}
catch (Exception ex)
{
    Console.WriteLine("Erro desconhecido!");
}
```



# Gerenciamento de erros (*try-catch*)

---

Outra maneira de escrever o mesmo try-catch:

```
try
{
    //BLOCO DE CÓDIGO SUPERVISIONADO
}
catch (IndexOutOfRangeException ex)
{
    Console.WriteLine(ex.ToString());
}
catch
{
    Console.WriteLine("Erro desconhecido!");
}
```





# Instruções verificadas

---

O que acontece se implementarmos o seguinte código?

```
int val1 = Int32.MaxValue;  
int val2;  
  
val2 = val1 + 1;  
  
Console.WriteLine("Valor2: " + val2);
```

# Instruções verificadas

---

O que acontece se implementarmos o seguinte código?

```
int val1 = Int32.MaxValue;  
int val2;  
  
val2 = val1 + 1;  
Console.WriteLine("Valor2: " + val2);
```

- a) Dispara exceção (erro no programa);
- b) Aparece: “Valor: -2147483648”
- c) Aparece: “Valor: 2147483647”
- d) Aparece: “Valor: 0”

# Instruções verificadas (*checked*)

---

O que acontece se implementarmos o seguinte código?

```
int val1 = Int32.MaxValue;  
int val2;  
  
val2 = val1 + 1;  
Console.WriteLine("Valor2: " + val2);
```

b) Aparece: “Valor: -2147483648”

# Instruções verificadas (*checked*)

---

No C#, quando temos uma aritmética de números inteiros, por padrão, não são lançadas exceções de estouro.

Esta implementação pode ajudar, no quesito “liberdade” de codificação (assim como é no C), porém pode te prejudicar, visto que o C# controla muito em outros aspectos (inicialização de variáveis).

Esta abordagem é padrão da linguagem (configurável no Visual Studio) e pode ser desativada.



# Instruções verificadas (*checked*)

---

Para desativá-la, vá no solution Explorer, clique com o botão direito do mouse na Application, entre em:

- Properties (ou Alt + Enter);
- Build;
- Clique no botão: “Advanced”;
- Marque a opção: “check for arithmetic overflow/underflow”

Recompile e execute novamente o código anterior.

# Instruções verificadas (*checked*)

---

E agora, o que acontece se implementarmos o seguinte código?

```
int val1 = Int32.MaxValue;  
int val2;  
  
val2 = val1 + 1;  
Console.WriteLine("Valor2: " + val2);
```

- a) Dispara exceção (erro no programa);
- b) Aparece: “Valor: -2147483648”
- c) Aparece: “Valor: 2147483647”
- d) Aparece: “Valor: 0”

# Instruções verificadas (*checked*)

---

E agora, o que acontece se implementarmos o seguinte código?

```
int val1 = Int32.MaxValue;  
int val2;  
  
val2 = val1 + 1;  
Console.WriteLine("Valor2: " + val2);
```

a) Dispara exceção (erro no programa);

# Instruções verificadas (*checked*)

---

Escopo de instruções verificadas:

*checked*

{

//BLOCO DE CÓDIGO COM ARITMÉTICA DE INTEIROS VERIFICADA

}

ou:

*checked*(EXPRESSÃO COM INTEIROS);





# Instruções não verificadas (*unchecked*)

---

Escopo de instruções não verificadas:

*unchecked*

```
{  
    //BLOCO DE CÓDIGO COM ARITMÉTICA DE INTEIROS VERIFICADA  
}
```

ou:

*unchecked*(EXPRESSÃO COM INTEIROS);



# Instruções verificadas (*checked*)

---

Apontamentos para instruções de aritmética verificadas:

- Não podem ser utilizadas em aritméticas de ponto-flutuante (não inteiros);
- São checadas aritméticas que estão diretamente no bloco *checked*. Caso haja uma chamada de método, a verificação não será encapsulada;
- O tipo Double não dispara exceções de overflow, nem quando se divide por 0 (zero). O .NET possui uma representação para infinito  
`Double.IsInfinity(double val);`

# Finalização de aula

---

- Não esqueçam de mandar as atividades práticas.
  - “Compactar o diretório da solution” e enviar no moodle.