



PROGRAMAÇÃO O.O.

(C#)



Se despedindo de vetores...
Professor: João Luiz Lagôas



Criando uma classe Carta



```
class Carta
{
    private string Face = "Ás";
    private string Naipe = "Espadas";

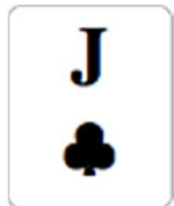
    public Carta(string f, string n)
    {
        if (face == "Ás" || face == "Dois" || face == "Três" || face == "Quatro" ||
            face == "Cinco" || face == "Seis" || face == "Sete" || face == "Oito" ||
            face == "Nove" || face == "Dez" || face == "Valete" || face == "Dama" ||
            face == "Rei")
            Face = f;

        if (naipe == "Espadas" || naipe == "Copas" ||
            naipe == "Paus" || naipe == "Ouros")
            Naipe = n;
    }

    public string Nome()
    {
        return Face + " de " + Naipe;
    }
}
```



Testando a classe Carta



```
class Program
{
    static void Main(string[] args)
    {
        Carta c1 = new Carta("Ás", "Copas");
        Carta c2 = new Carta("Dois", "Espadas");
        Carta c3 = new Carta("Valete", "Paus");
        Carta c4 = new Carta("Sete", "Ouros");

        Console.WriteLine(c1.Nome());
        Console.WriteLine(c2.Nome());
        Console.WriteLine(c3.Nome());
        Console.WriteLine(c4.Nome());

        Console.ReadLine();
    }
}
```

Diagram illustrating the mapping of the code to the cards:

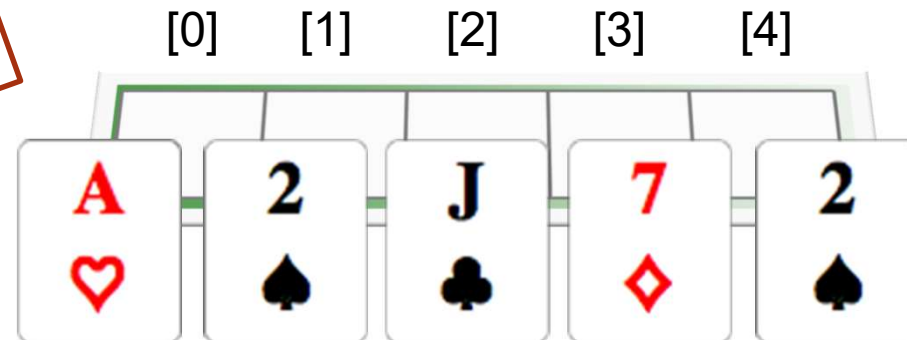
- `Carta c1 = new Carta("Ás", "Copas");` maps to the Ace of Hearts.
- `Carta c2 = new Carta("Dois", "Espadas");` maps to the Two of Spades.
- `Carta c3 = new Carta("Valete", "Paus");` maps to the Jack of Clubs.
- `Carta c4 = new Carta("Sete", "Ouros");` maps to the Seven of Diamonds.

E o baralho?



- Um baralho pode ser idealizado num primeiro momento como sendo uma array/vetor de cartas.

`Carta[] baralho = new Carta[5];`

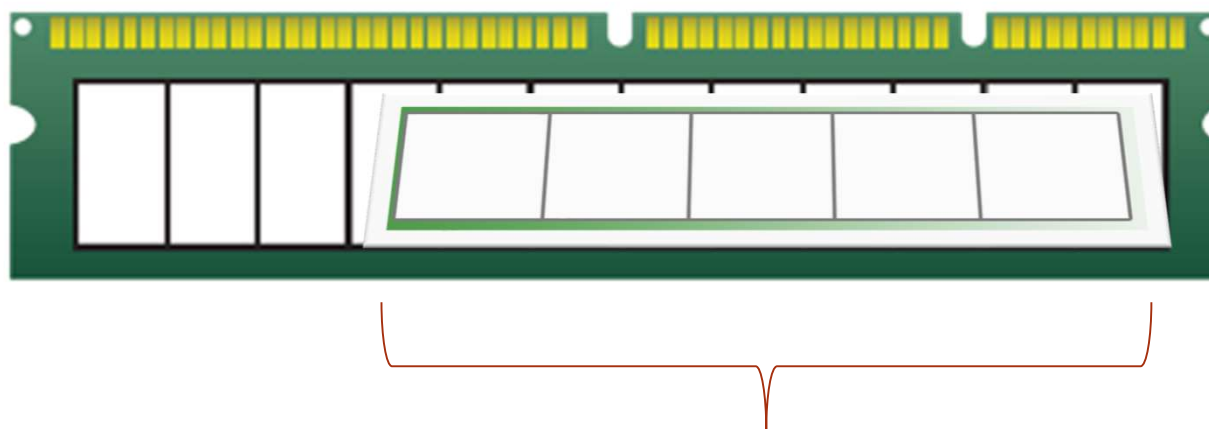


Vamos revisar o uso de vetores para determinar se construir um baralho como um vetor de objetos Carta é uma boa opção.

Array



- Uma array (ou vetor) é um grupo de posições de memória adjacentes que tem o mesmo nome e tipo.



Array ou vetor

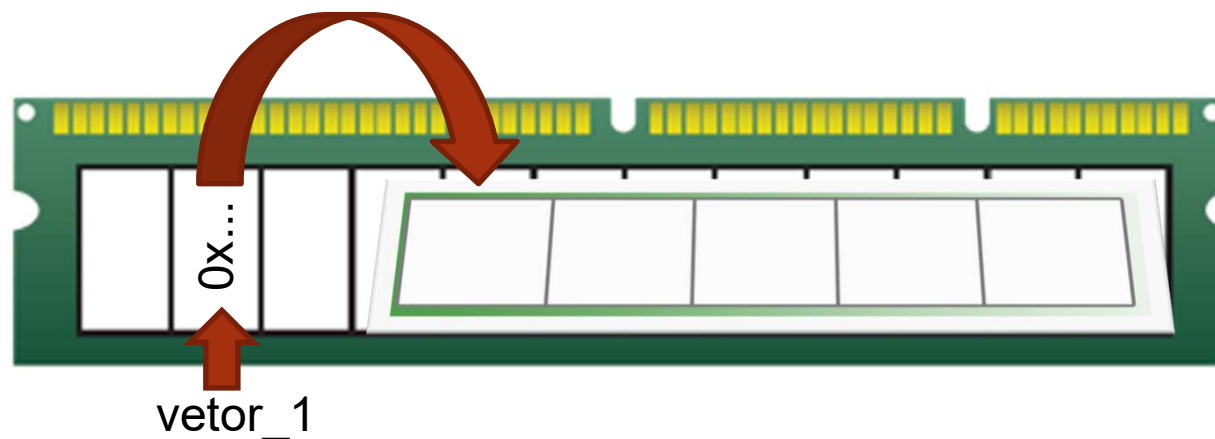
Uma array/vetor é um **objeto**! Não se esqueça disso!
E portanto um **tipo por referência**!

Declarando e Criando uma array

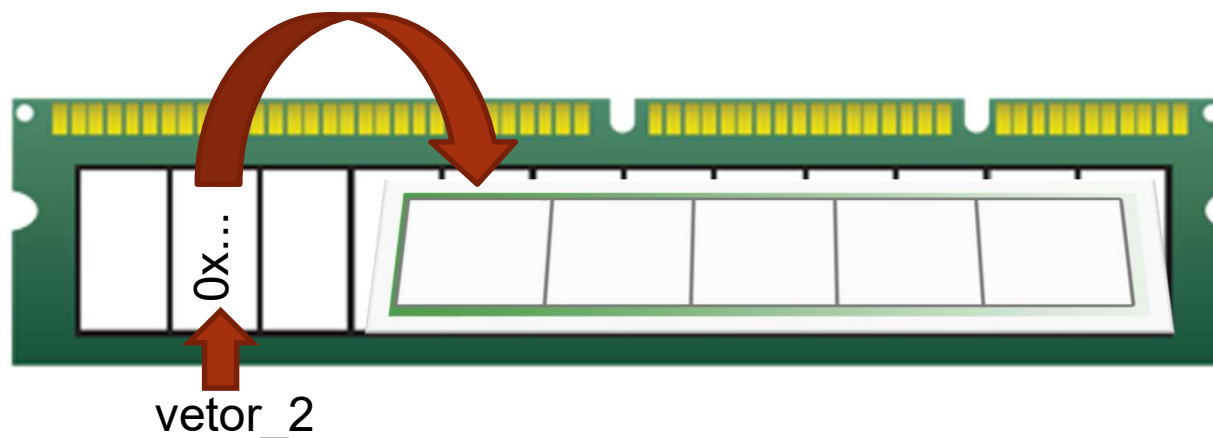


Podemos criar uma array de duas formas diferentes, mas em ambos os casos **devemos determinar o seu tamanho**.

```
int[] vetor_1 = new int[5];
```



```
int[] vetor_2 = { 1, 4, 1, 2, 4 };
```



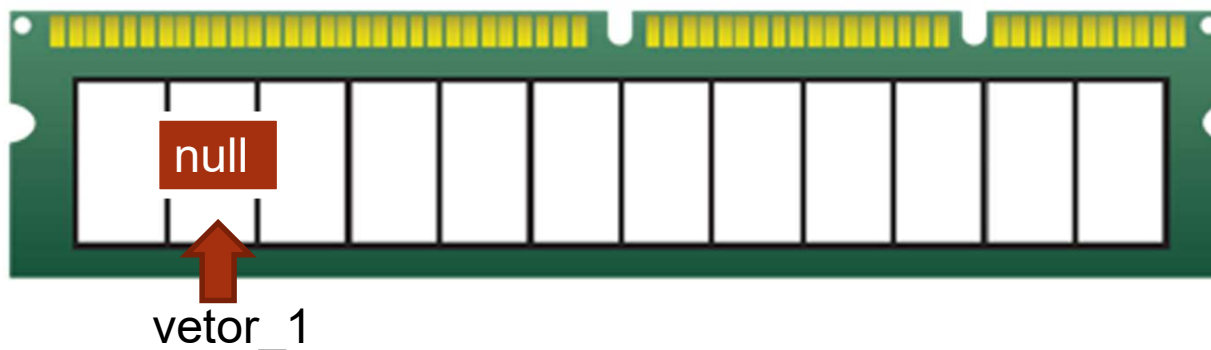
Declarando e Criando uma array



```
int[] vetor_1;
```



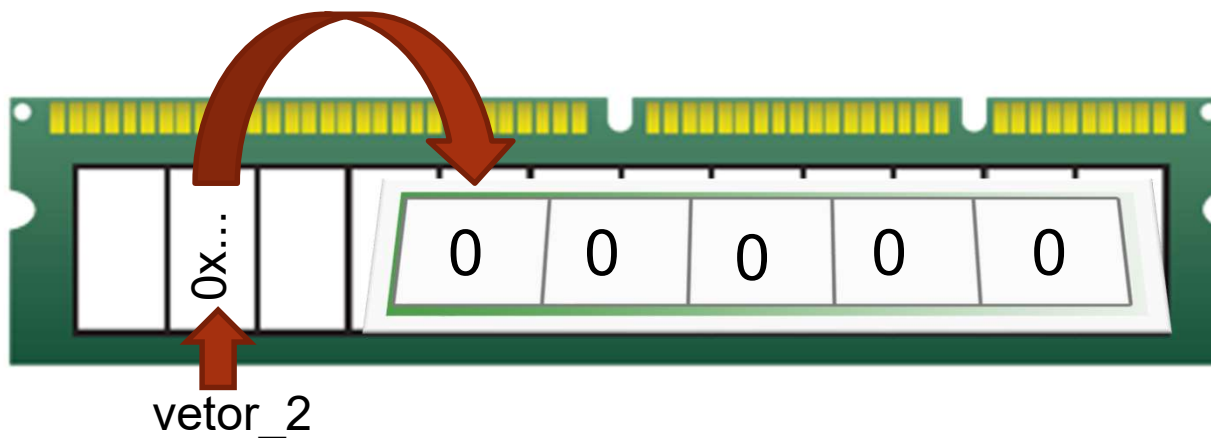
VARIÁVEL QUE É DO TIPO VETOR DE INTEIROS.



```
vetor_1 = new int[5];
```



CRIACAO DO VETOR NA MEMORIA

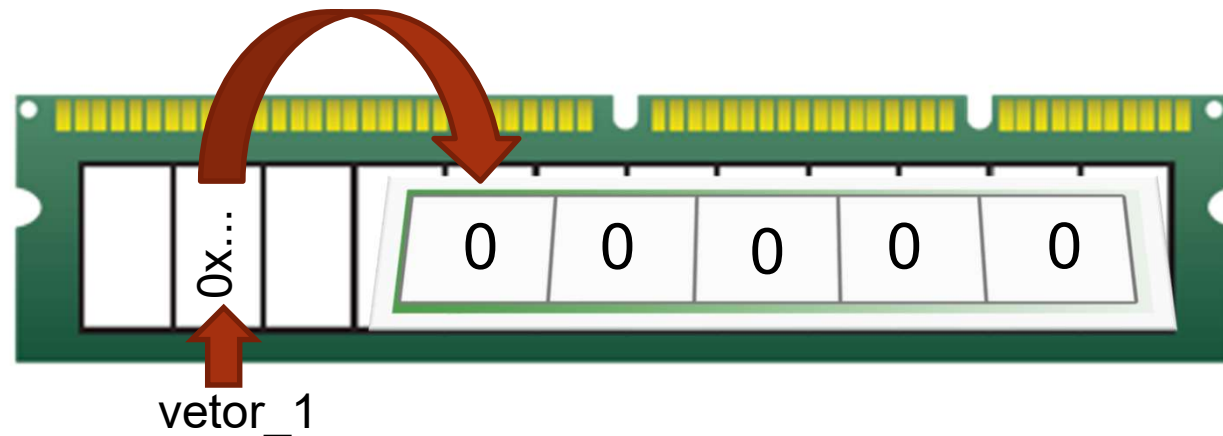


Valores default de uma array



- Quando os arrays são alocados, os elementos são inicializados com zero para variáveis numéricas, com false para variáveis booleanas e null para variáveis por referência.

```
int[] vetor_1 = new int[5];
```



Tamanho da array



- Toda array em C# conhece seu próprio tamanho, basta escrever o comando `nome_da_array.Length`.

```
int[] vetor_1 = new int[5];
```

```
Console.WriteLine(vetor_1.Length);
```

Irá imprimir 5 no Console

Acessando os elementos



- Para nos referirmos a uma posição ou elemento particular na array, especificamos o nome da array e um **índice**.

```
int[] vetor_1 = new int[5];
```

```
vetor_1[0] = 3;
```

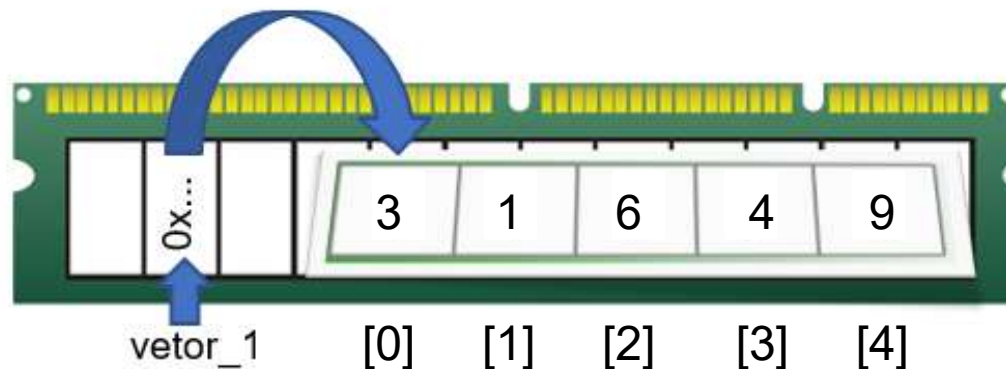
```
vetor_1[1] = 1;
```

```
vetor_1[2] = 6;
```

```
vetor_1[3] = 4;
```

```
vetor_1[4] = 9;
```

```
Console.WriteLine(vetor_1[2]);
```



Acessando os elementos



- Se uma posição de memória for acessada no vetor mas nunca foi alocada (ultrapassar os limites da array), uma **exceção** será gerada.

```
int[] vetor_1 = new int[5];
```

```
vetor_1[0] = 3;
```

```
vetor_1[1] = 1;
```

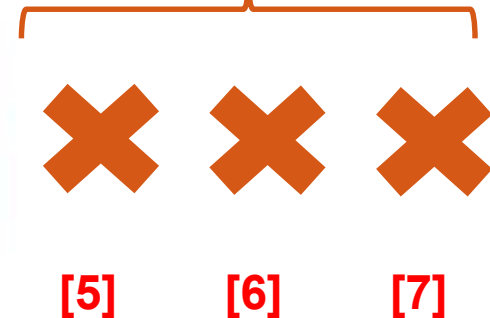
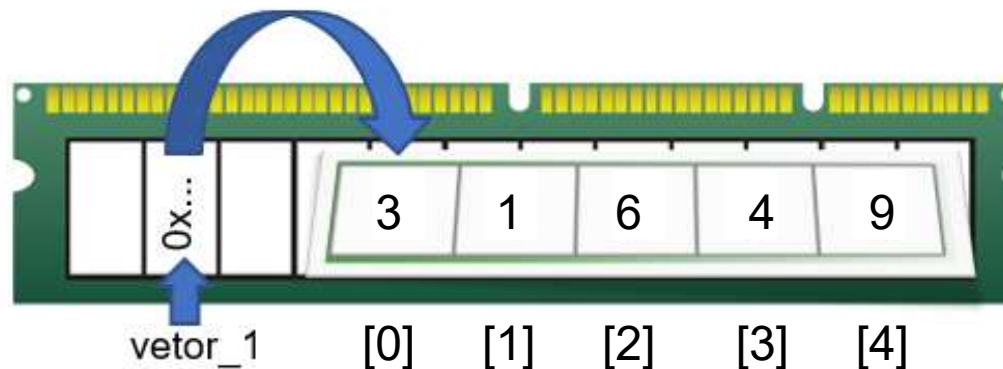
```
vetor_1[2] = 6;
```

```
vetor_1[3] = 4;
```

```
vetor_1[4] = 9;
```

```
Console.WriteLine(vetor_1[7]);
```

Essas posições nunca foram alocadas na criação do vetor!

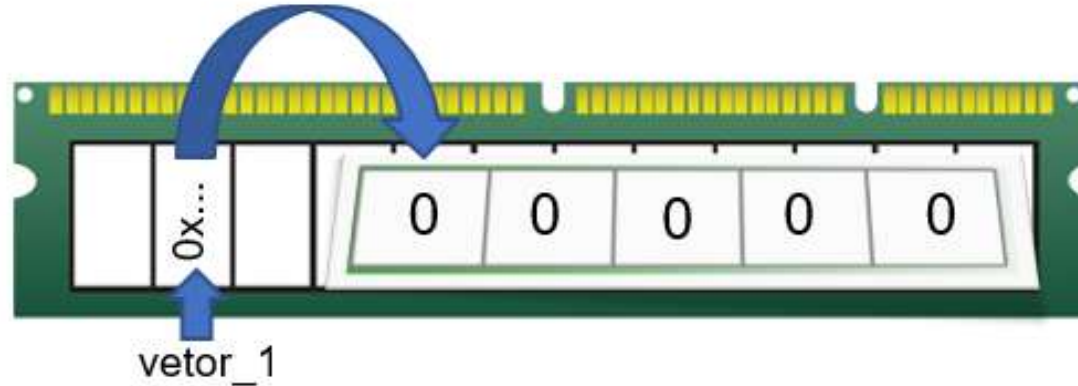


Arrays de tipos por referência



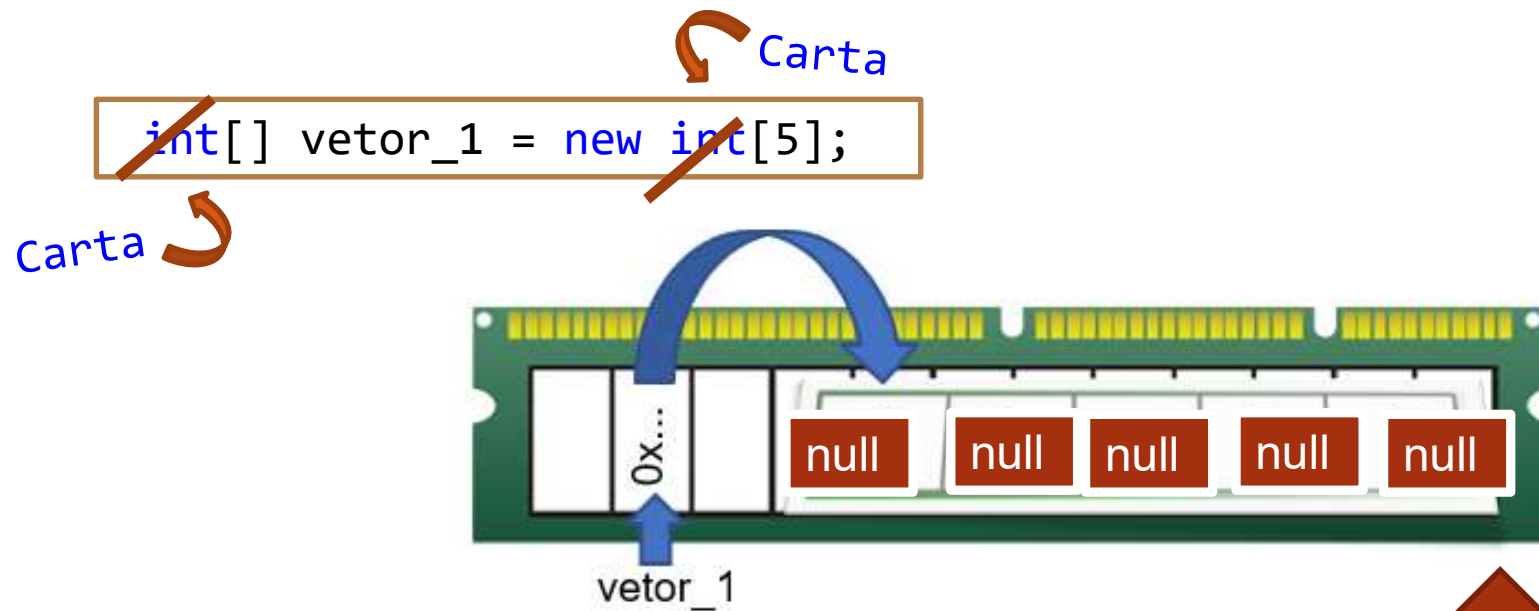
- Em cada posição da array, já sabemos que podemos armazenar valores.

```
int[] vetor_1 = new int[5];
```



- Neste exemplo, armazenamos inteiros em cada posição de memória.

Arrays de tipos por referência



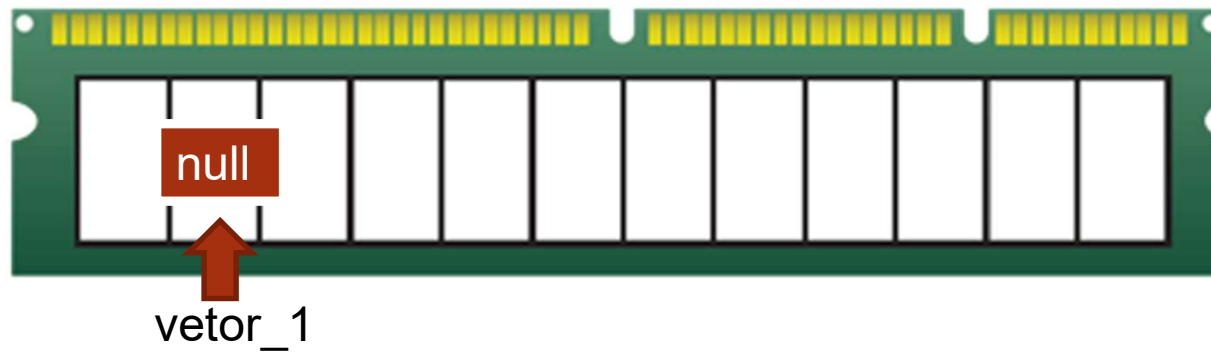
No entanto, podemos armazenar objetos (tipos por referência) em cada posição do vetor também. O raciocínio é o mesmo.

Arrays de tipos por referência

```
Carta[] vetor_1;
```



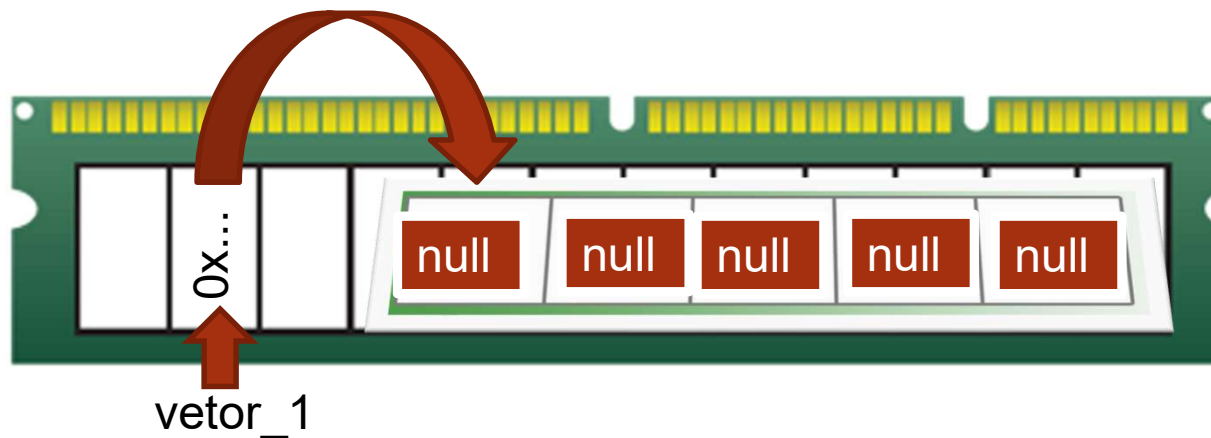
VARIÁVEL QUE É DO TIPO VETOR DE CARTAS.



```
vetor_1 = new int[5];
```



CRIACAO DO VETOR NA MÉMORIA



Arrays de tipos por referência



- Preenchendo o vetor com objetos do tipo **Carta**.

```
Carta[] vetor_1 = new Carta[5];  
  
Carta c1 = new Carta("Ás", "Copas");  
Carta c2 = new Carta("Dois", "Espadas");  
Carta c3 = new Carta("Valete", "Paus");  
Carta c4 = new Carta("Sete", "Ouros");  
  
vetor_1[0] = c1;  
vetor_1[1] = c2;  
vetor_1[2] = c3;  
vetor_1[3] = c4;  
vetor_1[4] = null;
```

Arrays de tipos por referência

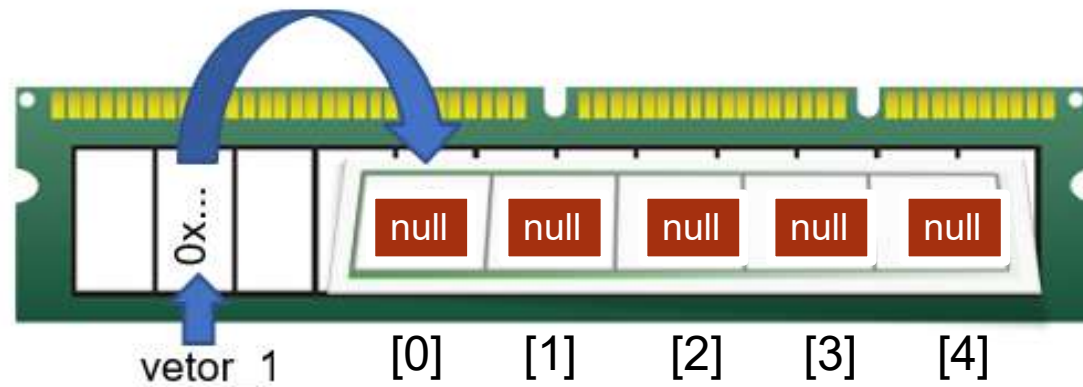


- Preenchendo o vetor com objetos do tipo **Carta**.

```
Carta[] vetor_1 = new Carta[5];
```

```
Carta c1 = new Carta("Ás", "Copas");  
Carta c2 = new Carta("Dois", "Espadas");  
Carta c3 = new Carta("Valete", "Paus");  
Carta c4 = new Carta("Sete", "Ouros");
```

```
vetor_1[0] = c1;  
vetor_1[1] = c2;  
vetor_1[2] = c3;  
vetor_1[3] = c4;  
vetor_1[4] = null;
```



Arrays de tipos por referência

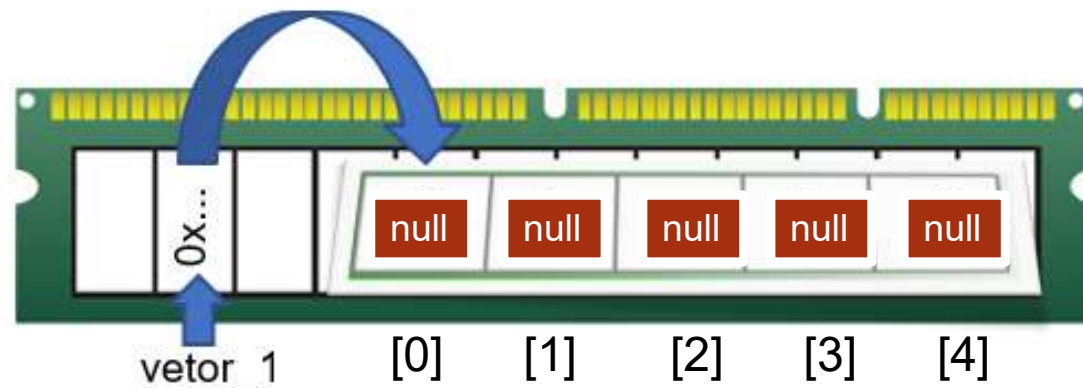


- Preenchendo o vetor com objetos do tipo **Carta**.

```
Carta[] vetor_1 = new Carta[5];
```

```
Carta c1 = new Carta("Ás", "Copas");  
Carta c2 = new Carta("Dois", "Espadas");  
Carta c3 = new Carta("Valete", "Paus");  
Carta c4 = new Carta("Sete", "Ouros");
```

```
vetor_1[0] = c1;  
vetor_1[1] = c2;  
vetor_1[2] = c3;  
vetor_1[3] = c4;  
vetor_1[4] = null;
```



Arrays de tipos por referência

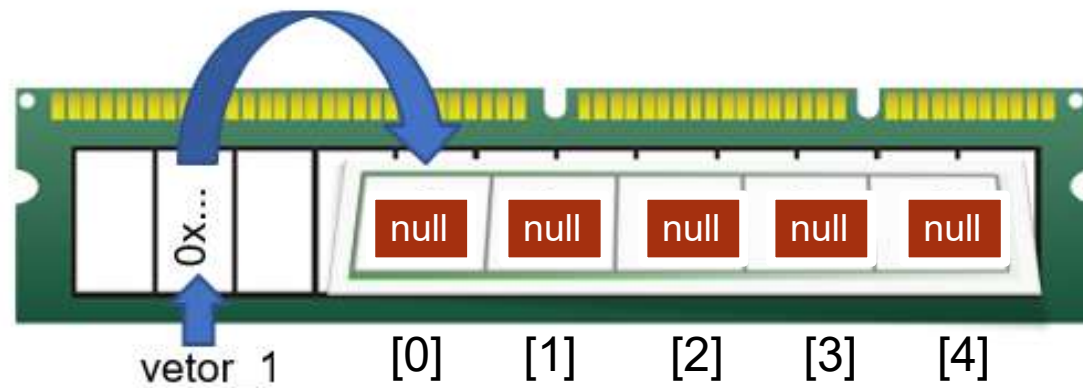


- Preenchendo o vetor com objetos do tipo **Carta**.

```
Carta[] vetor_1 = new Carta[5];
```

```
Carta c1 = new Carta("Ás", "Copas");  
Carta c2 = new Carta("Dois", "Espadas");  
Carta c3 = new Carta("Valete", "Paus");  
Carta c4 = new Carta("Sete", "Ouros");
```

```
vetor_1[0] = c1;  
vetor_1[1] = c2;  
vetor_1[2] = c3;  
vetor_1[3] = c4;  
vetor_1[4] = null;
```



Arrays de tipos por referência



- Preenchendo o vetor com objetos do tipo **Carta**.

```
Carta[] vetor_1 = new Carta[5];
```

```
Carta c1 = new Carta("Ás", "Copas");
```

```
Carta c2 = new Carta("Dois", "Espadas");
```

```
Carta c3 = new Carta("Valete", "Paus");
```

```
Carta c4 = new Carta("Sete", "Ouros");
```

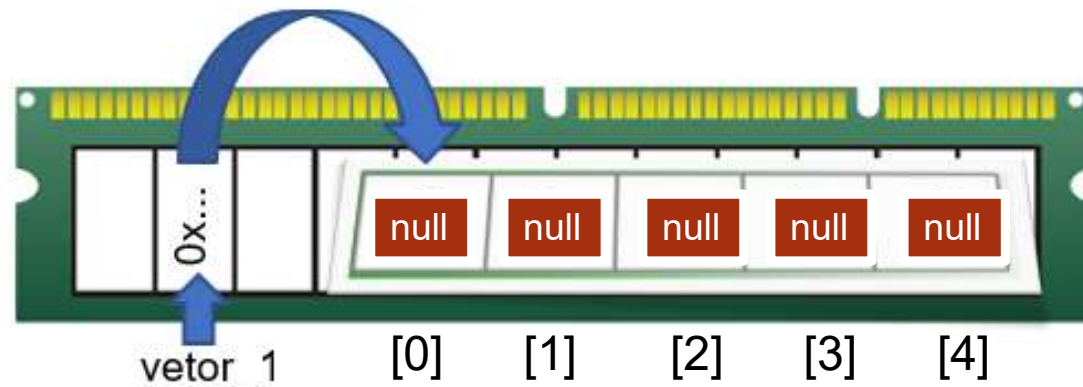
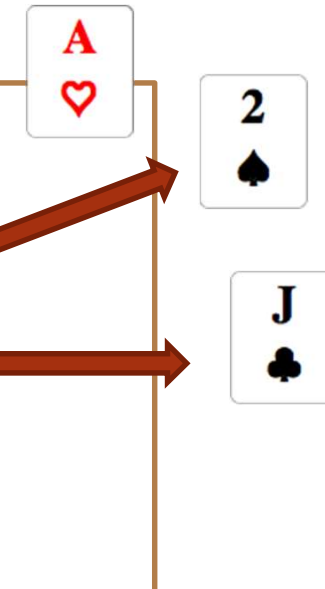
```
vetor_1[0] = c1;
```

```
vetor_1[1] = c2;
```

```
vetor_1[2] = c3;
```

```
vetor_1[3] = c4;
```

```
vetor_1[4] = null;
```



Arrays de tipos por referência



- Preenchendo o vetor com objetos do tipo **Carta**.

```
Carta[] vetor_1 = new Carta[5];
```

```
Carta c1 = new Carta("Ás", "Copas");
```

```
Carta c2 = new Carta("Dois", "Espadas");
```

```
Carta c3 = new Carta("Valete", "Paus");
```

```
Carta c4 = new Carta("Sete", "Ouros");
```

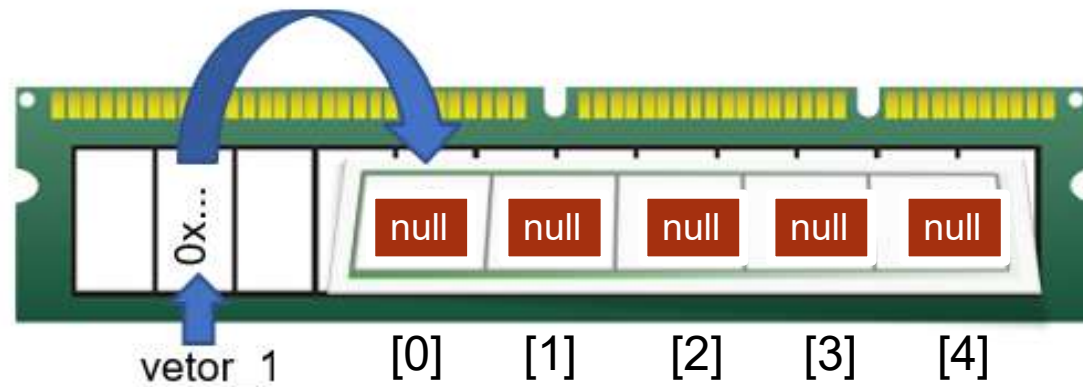
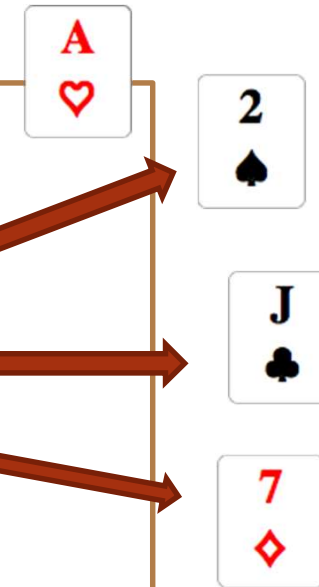
```
vetor_1[0] = c1;
```

```
vetor_1[1] = c2;
```

```
vetor_1[2] = c3;
```

```
vetor_1[3] = c4;
```

```
vetor_1[4] = null;
```



Arrays de tipos por referência



- Preenchendo o vetor com objetos do tipo **Carta**.

```
Carta[] vetor_1 = new Carta[5];
```

```
Carta c1 = new Carta("Ás", "Copas");
```

```
Carta c2 = new Carta("Dois", "Espadas");
```

```
Carta c3 = new Carta("Valete", "Paus");
```

```
Carta c4 = new Carta("Sete", "Ouros");
```

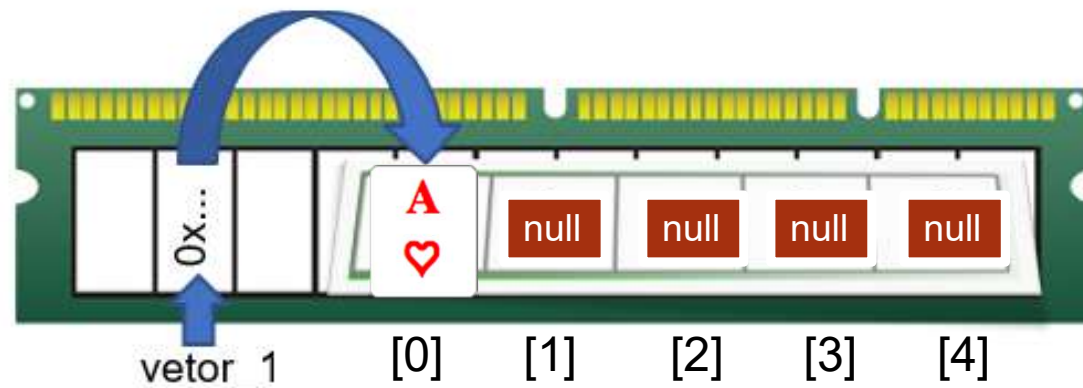
```
vetor_1[0] = c1;
```

```
vetor_1[1] = c2;
```

```
vetor_1[2] = c3;
```

```
vetor_1[3] = c4;
```

```
vetor_1[4] = null;
```



Arrays de tipos por referência



- Preenchendo o vetor com objetos do tipo **Carta**.

```
Carta[] vetor_1 = new Carta[5];
```

```
Carta c1 = new Carta("Ás", "Copas");
```

```
Carta c2 = new Carta("Dois", "Espadas");
```

```
Carta c3 = new Carta("Valete", "Paus");
```

```
Carta c4 = new Carta("Sete", "Ouros");
```

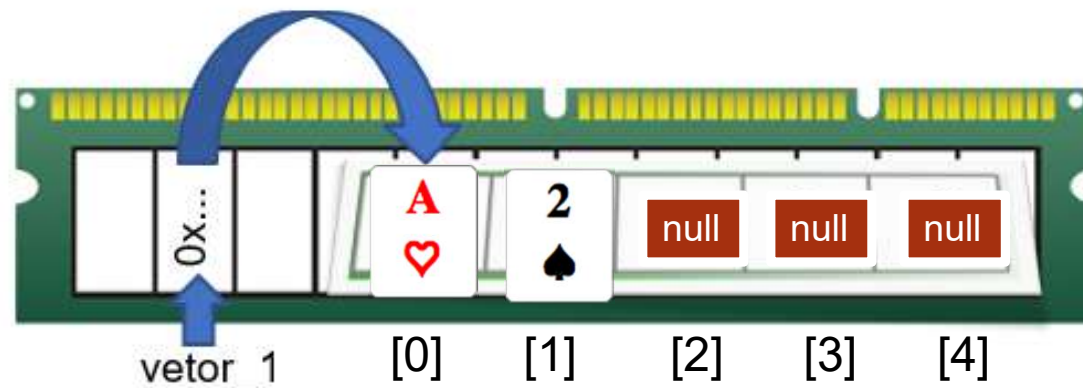
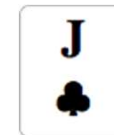
```
vetor_1[0] = c1;
```

```
vetor_1[1] = c2;
```

```
vetor_1[2] = c3;
```

```
vetor_1[3] = c4;
```

```
vetor_1[4] = null;
```



Arrays de tipos por referência



- Preenchendo o vetor com objetos do tipo **Carta**.

```
Carta[] vetor_1 = new Carta[5];
```

```
Carta c1 = new Carta("Ás", "Copas");
```

```
Carta c2 = new Carta("Dois", "Espadas");
```

```
Carta c3 = new Carta("Valete", "Paus");
```

```
Carta c4 = new Carta("Sete", "Ouros");
```

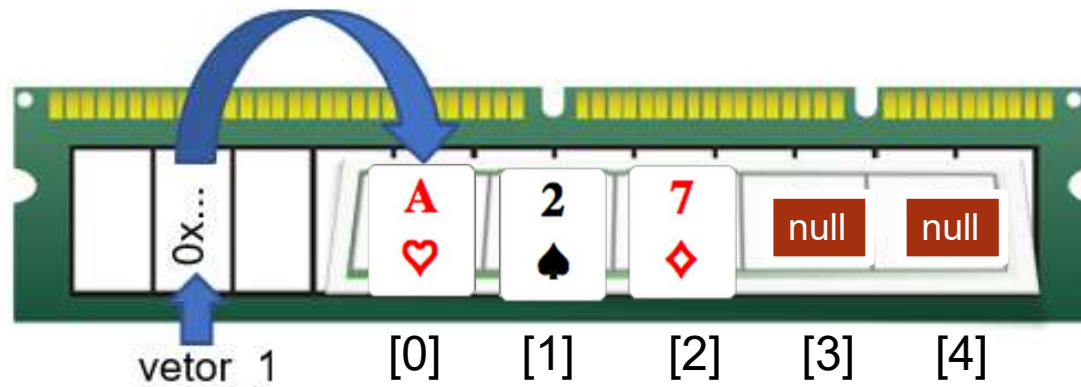
```
vetor_1[0] = c1;
```

```
vetor_1[1] = c2;
```

```
vetor_1[2] = c3;
```

```
vetor_1[3] = c4;
```

```
vetor_1[4] = null;
```



Arrays de tipos por referência



- Preenchendo o vetor com objetos do tipo **Carta**.

```
Carta[] vetor_1 = new Carta[5];
```

```
Carta c1 = new Carta("Ás", "Copas");
```

```
Carta c2 = new Carta("Dois", "Espadas");
```

```
Carta c3 = new Carta("Valete", "Paus");
```

```
Carta c4 = new Carta("Sete", "Ouros");
```

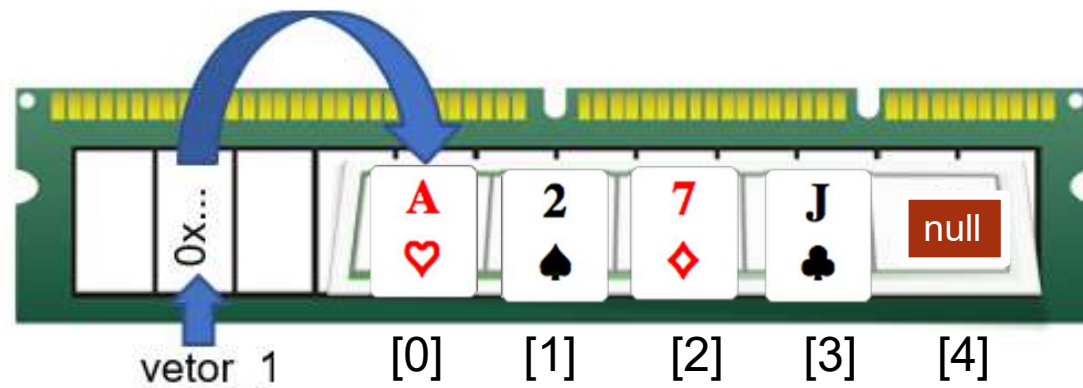
```
vetor_1[0] = c1;
```

```
vetor_1[1] = c2;
```

```
vetor_1[2] = c3;
```

```
vetor_1[3] = c4;
```

```
vetor_1[4] = null;
```



Arrays de tipos por referência



- Preenchendo o vetor com objetos do tipo **Carta**.

```
Carta[] vetor_1 = new Carta[5];
```

```
Carta c1 = new Carta("Ás", "Copas");
```

```
Carta c2 = new Carta("Dois", "Espadas");
```

```
Carta c3 = new Carta("Valete", "Paus");
```

```
Carta c4 = new Carta("Sete", "Ouros");
```

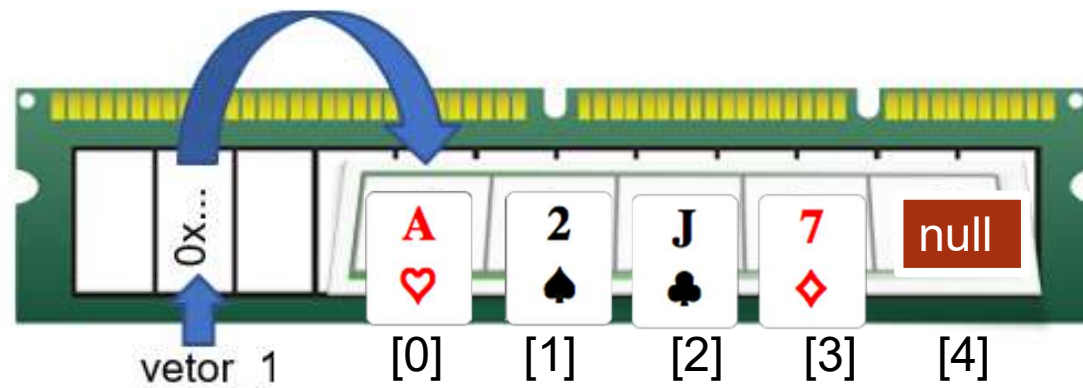
```
vetor_1[0] = c1;
```

```
vetor_1[1] = c2;
```

```
vetor_1[2] = c3;
```

```
vetor_1[3] = c4;
```

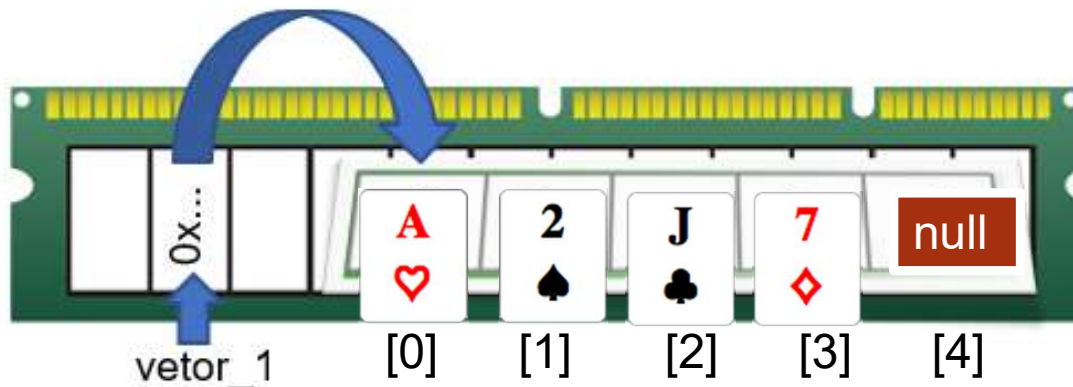
```
vetor_1[4] = null;
```



Acessando os elementos

- O que será impresso no Console?

```
Console.WriteLine(vetor_1[2].Nome());
```



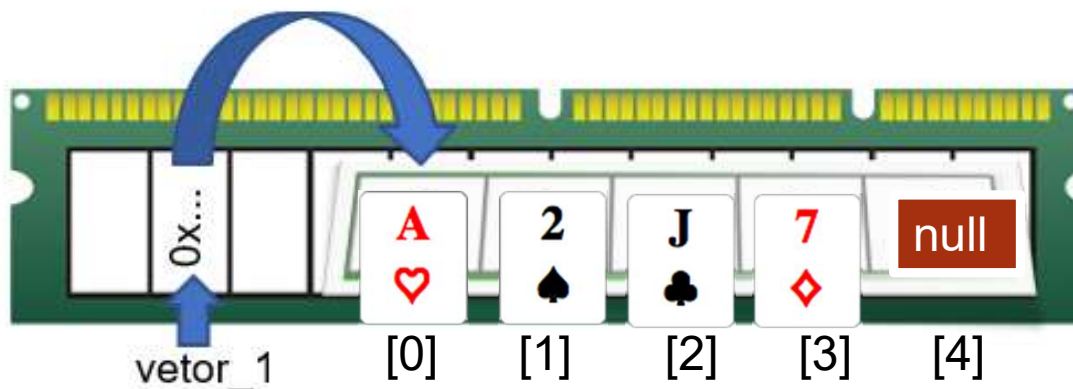
Acessando os elementos

- O que será impresso no Console?

```
Console.WriteLine(vetor_1[2].Nome());
```

Console

“Valete de Paus”



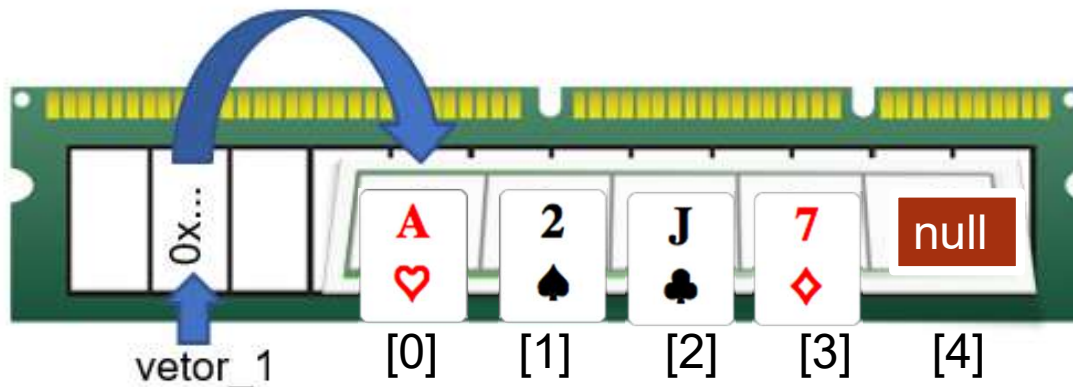
Resposta

Acessando os elementos

- Se em cada posição do vetor há um objeto Carta, então eu posso acessar este objeto através de um índice.

```
Console.WriteLine(vetor_1[2].Nome());
```

Carta[]



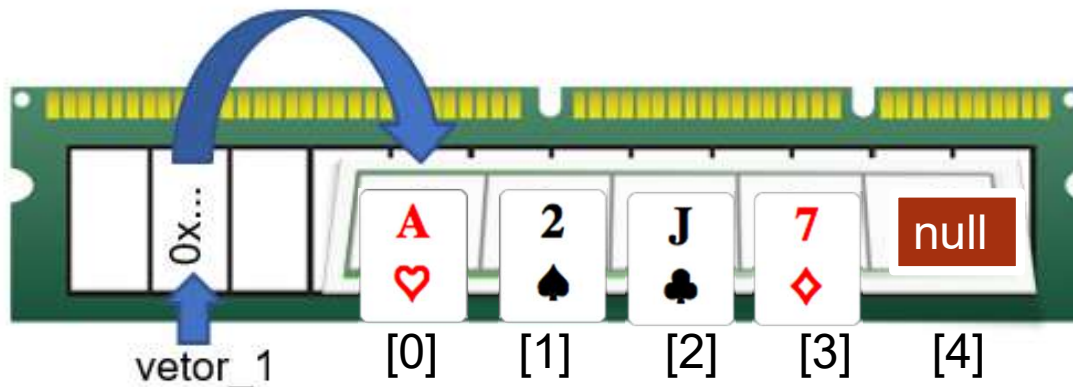
Acessando os elementos

- Se em cada posição do vetor há um objeto Carta, então eu posso acessar este objeto através de um índice.

```
Console.WriteLine(vetor_1[2].Nome());
```

Carta[]

Carta



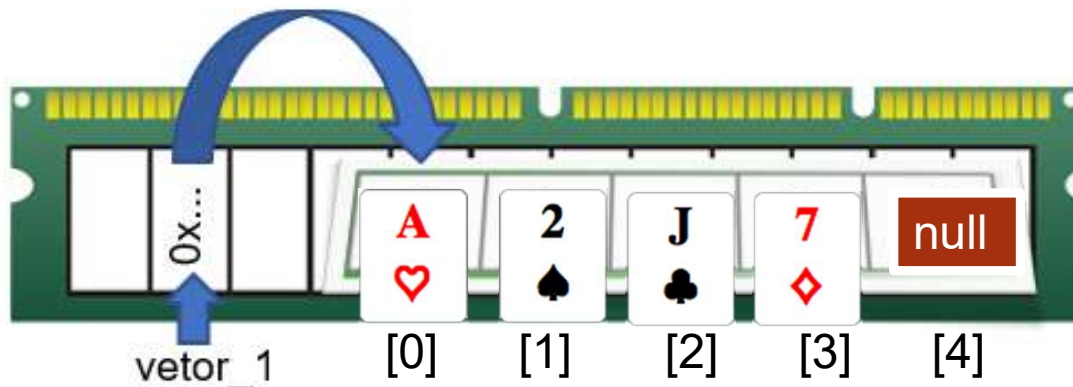
Acessando os elementos

- Se em cada posição do vetor há um objeto Carta, então eu posso acessar este objeto através de um índice.

```
Console.WriteLine(vetor_1[2].Nome());
```

Carta[]

Carta

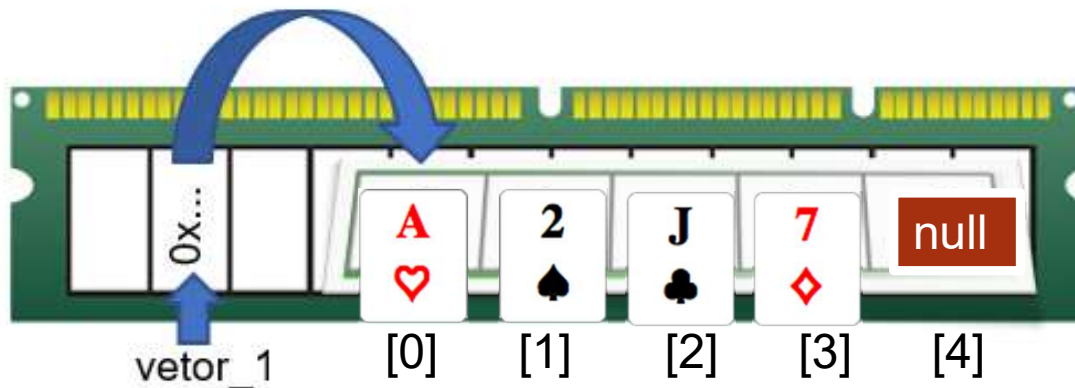


Método de um
objeto Carta que
retorna uma **string**!

Acessando os elementos

- Obviamente podemos utilizar um for para percorrer o vetor como sempre fizemos...

```
for (int i = 0; i < 4; i++)  
    Console.WriteLine(vetor_1[i].Nome());
```



Duas perguntas!

- E se o vetor de **Cartas** precisar **armazenar mais cartas**... O que fazer??
- E se for necessário **remover** cartas do vetor de **Cartas**... O que fazer??



Armazenando mais dados do que foi alocado



- Se o programador necessitar aumentar o tamanho do vetor de modo a armazenar mais dados?

```
Carta[] vetor_1 = new Carta[5];  
  
Carta c1 = new Carta("Ás", "Copas");  
Carta c2 = new Carta("Dois", "Espadas");  
Carta c3 = new Carta("Valete", "Paus");  
Carta c4 = new Carta("Sete", "Ouros");  
  
vetor_1[0] = c1;  
vetor_1[1] = c2;  
vetor_1[2] = c3;  
vetor_1[3] = c4;  
vetor_1[4] = null;
```

- Por exemplo, dado o código acima, se o programador agora quiser um vetor para armazenar 10 Cartas, como proceder?

Armazenando mais dados do que foi alocado



- Se o programador necessitar aumentar o tamanho do vetor de modo a armazenar mais dados?

```
Carta[] vetor_1 = new Carta[5];
```

```
Carta c1 = new Carta("Ás", "Copas");  
Carta c2 = new Carta("Dois", "Espadas");  
Carta c3 = new Carta("Valete", "Paus");  
Carta c4 = new Carta("Sete", "Ouros");
```

```
vetor_1[0] = c1;  
vetor_1[1] = c2;  
vetor_1[2] = c3;  
vetor_1[3] = c4;  
vetor_1[4] = null;
```



**A solução é criar um novo vetor de tamanho 10 e copiar os elementos do vetor de tamanho 5 para o novo.
Não parece adequado!**

- Por exemplo, dado o código acima, se o programador agora quiser um vetor para armazenar 10 Cartas, como proceder?

Removendo dados de um vetor



- Se o programador necessitar remover dados do vetor e desalocar a memória para a posição do elemento?

```
Carta[] vetor_1 = new Carta[5];  
  
Carta c1 = new Carta("Ás", "Copas");  
Carta c2 = new Carta("Dois", "Espadas");  
Carta c3 = new Carta("Valete", "Paus");  
Carta c4 = new Carta("Sete", "Ouros");  
  
vetor_1[0] = c1;  
vetor_1[1] = c2;  
vetor_1[2] = c3;  
vetor_1[3] = c4;  
vetor_1[4] = null;
```

- Por exemplo, dado o código acima, se o programador agora quiser retirar a carta c3 da posição [2] do vetor?

Removendo dados de um vetor



- Se o programador necessitar remover dados do vetor e desalocar a memória para a posição do elemento?

```
Carta[] vetor_1 = new Carta[5];
```

```
Carta c1 = new Carta("Ás", "Copas");
```

```
Carta c2 = new Carta("Dois", "Espadas");
```

```
Carta c3 = new Carta("Valete", "Paus");
```

```
Carta c4 = new Carta("Sete", "Ouros");
```

```
vetor_1[0] = c1;
```

```
vetor_1[1] = c2;
```

```
vetor_1[2] = c3;
```

```
vetor_1[3] = c4;
```

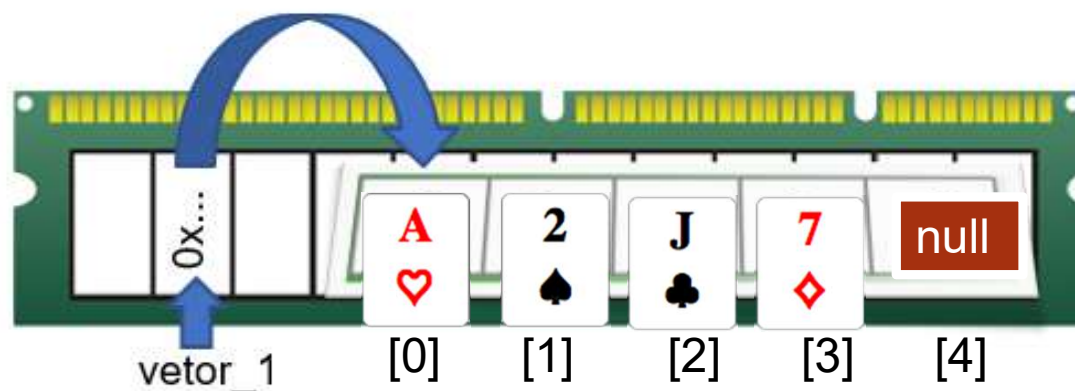
```
vetor_1[4] = null;
```

Basta escrever o código abaixo?

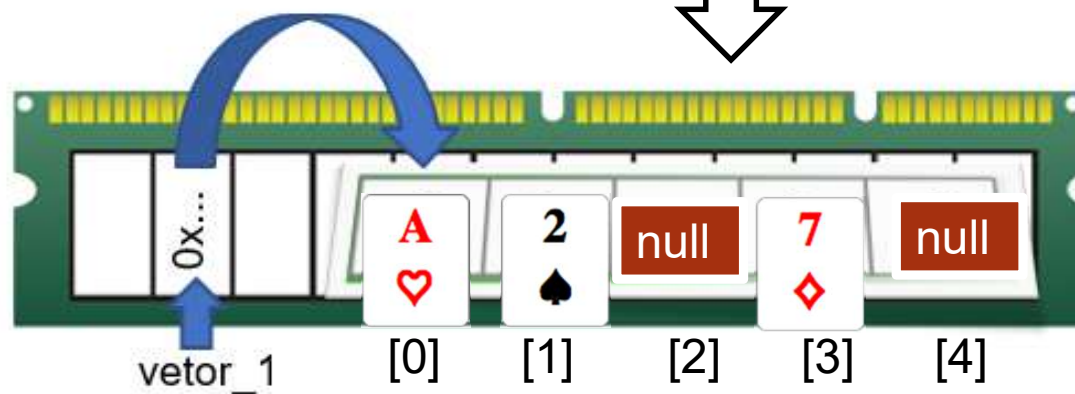
```
Vetor_1[2] = null;
```

- Por exemplo, dado o código acima, se o programador agora quiser retirar a carta c3 da posição [2] do vetor?

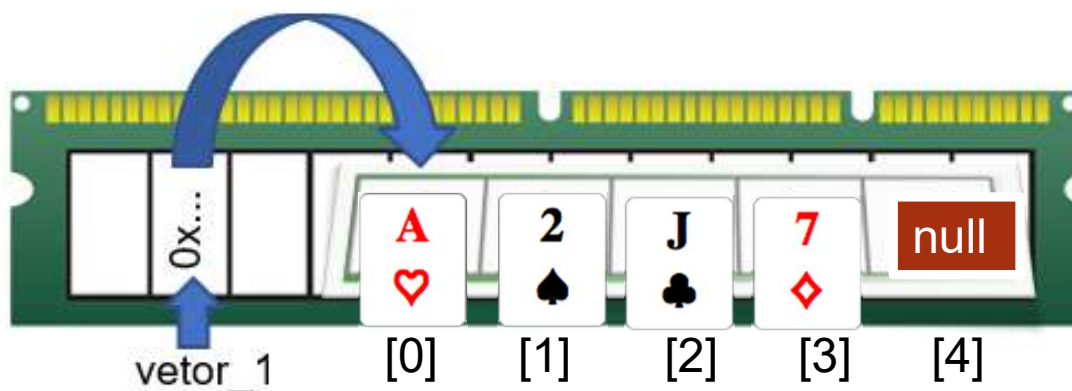
Removendo dados de um vetor



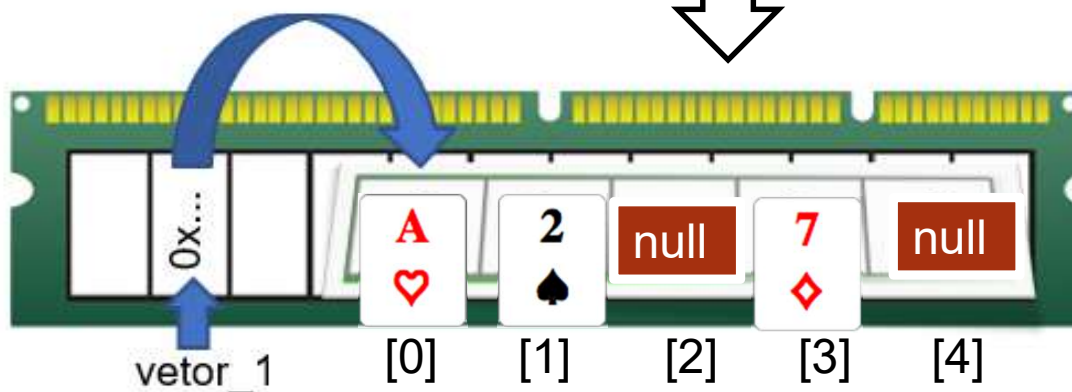
Vetor_1[2] = null;



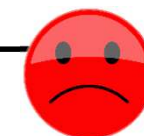
Removendo dados de um vetor



Vetor_1[2] = null;



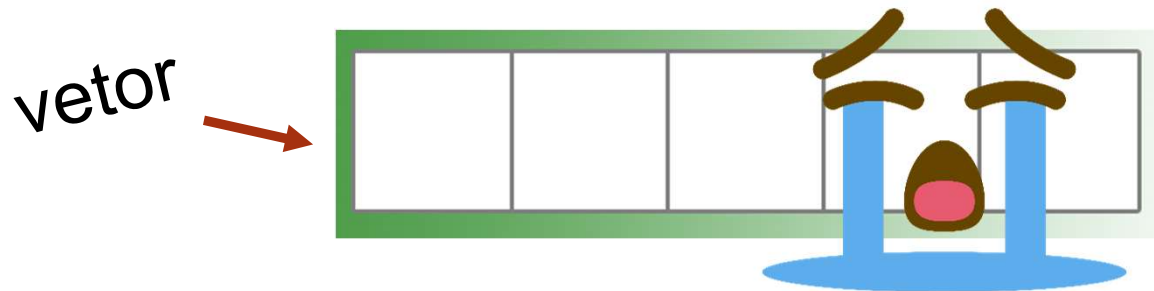
Repare que a posição [2] ainda está presente no vetor e o vetor ainda tem tamanho 5!



Problemas com vetores...



- Apesar de ser comumente usada, arrays/vetores têm algumas limitações.
- **O tamanho deve ser especificado e é fixo na hora de se criar um novo vetor;**
- **Não é possível remover elementos do vetor e eliminar a alocação da posição de memória.**



Coleções







- C# provê várias classes conhecidas como *collections*.
- Essas classes são usadas para armazenar conjuntos de dados de um mesmo tipo (assim como arrays).
- A vantagem de se utilizar classes *collections* é que muitas delas **resolvem problemas que existem em vetores entre outras características positivas.**

Coleções



- O Framework .NET tem diversas classes de coleções que lidam com as situações complicadas encontradas pelo uso de vetores.
- Algumas dessas classes de coleções são:

	Queue	Representa uma coleção em formato de fila. O primeira a entrar é o primeiro a sair.
	Stack	Representa uma coleção em formato de pilha. O última a entrar é o primeiro a sair.
	HashTable	Representa uma coleção de pares chave-valor organizados com base no código hash da chave.
	List	Representa uma lista fortemente tipada de objetos que podem ser acessados por índice.

namespace
Collections

Listas ao invés de Arrays/Vetores



- A classe mais popular e comum das coleções C# é denominada **List**.
- Um objeto do tipo List é semelhante com um objeto do tipo Array (vetor): é usado para armazenar vários elementos de dados.
- No entanto, objetos List apresentam diversas facilidades que não são encontradas em vetores.
- Uma vez que você cria um objeto do tipo List, é **fácil** adicionar, remover, observar e mesmo mover itens de um lugar da lista para o outro.



Exemplo



- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

```
List<int> minhaLista = new List<int>();
```

```
for (int i = 0; i < 3; i++)  
    minhaLista.Add(i * 2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
minhaLista.RemoveAt(2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
Console.ReadLine();
```

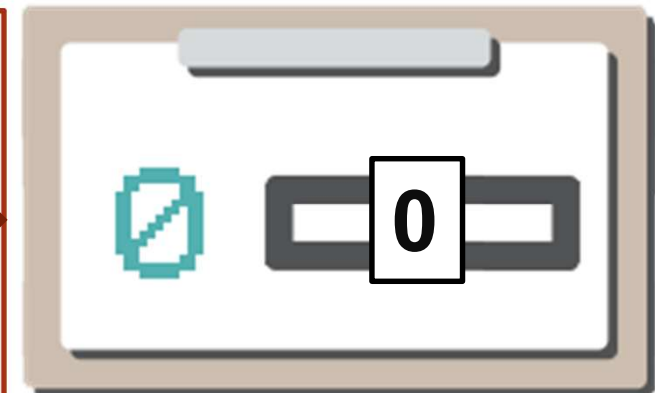


Exemplo



- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i < 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```

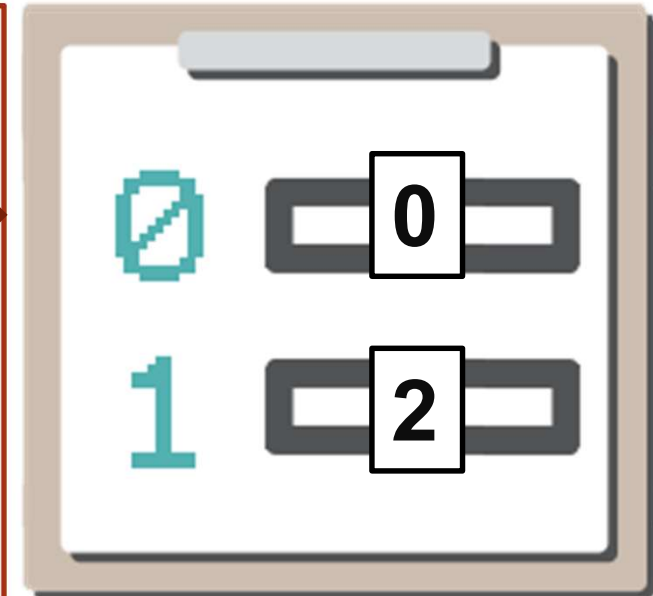


Exemplo



- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

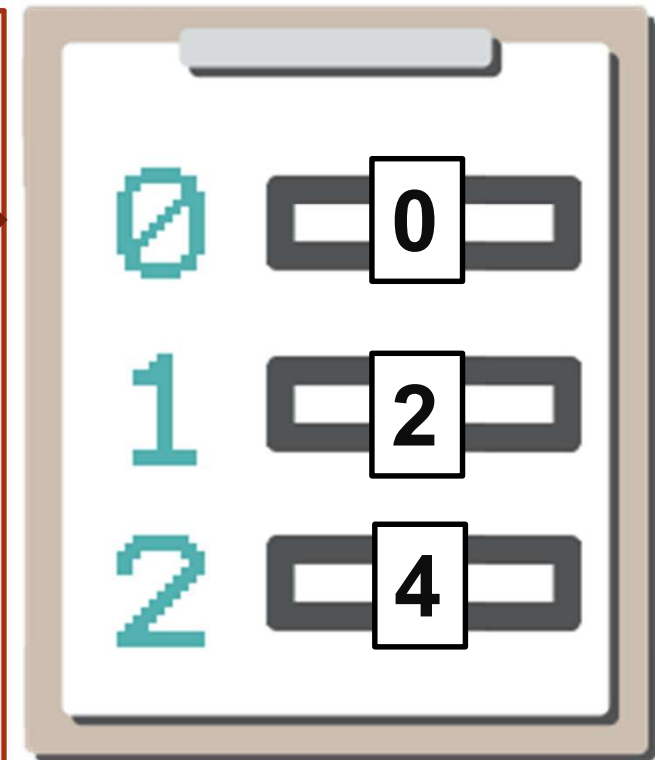
```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i < 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```



Exemplo

- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i < 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```

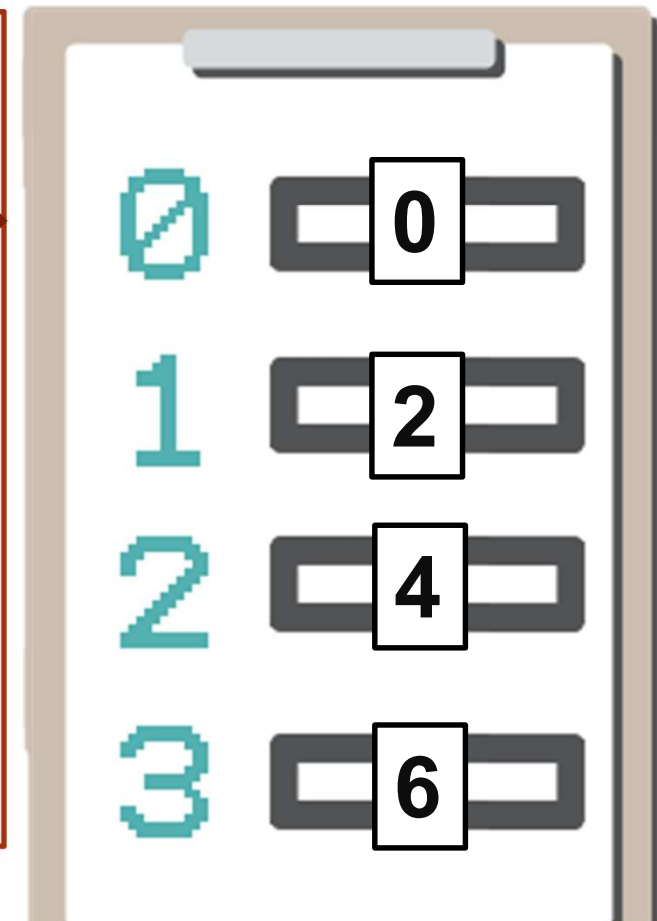


Exemplo



- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i < 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```



Exemplo

Console

0



```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i < 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```

0	0
1	2
2	4
3	6

Exemplo

Console

```
0  
2
```

```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i < 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```



0	0
1	2
2	4
3	6

Exemplo



Console

0
2
4

List<int

```
for (int i = 0; i < 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```

0	0
1	2
2	4
3	6

Exemplo



Console

```
0
2
4
6
```

List<int

```
for (int i = 0; i < 5; i++)
    minhaLista.Add(i * 2);

for (int i = 0; i < minhaLista.Count; i++)
    Console.WriteLine(minhaLista[i]);

minhaLista.RemoveAt(2);

for (int i = 0; i < minhaLista.Count; i++)
    Console.WriteLine(minhaLista[i]);

Console.ReadLine();
```

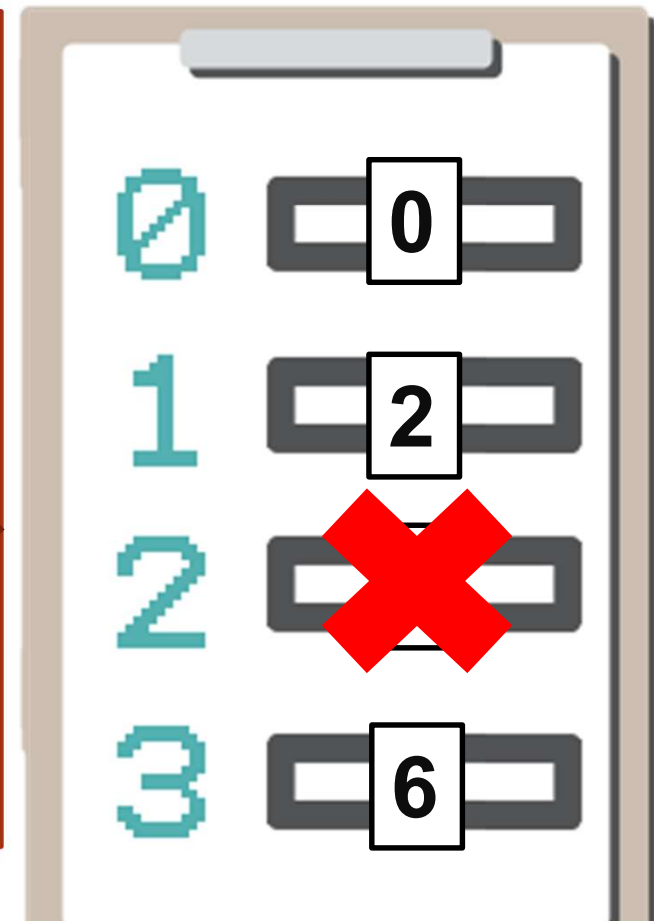
0	0
1	2
2	4
3	6

Exemplo



- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i < 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```

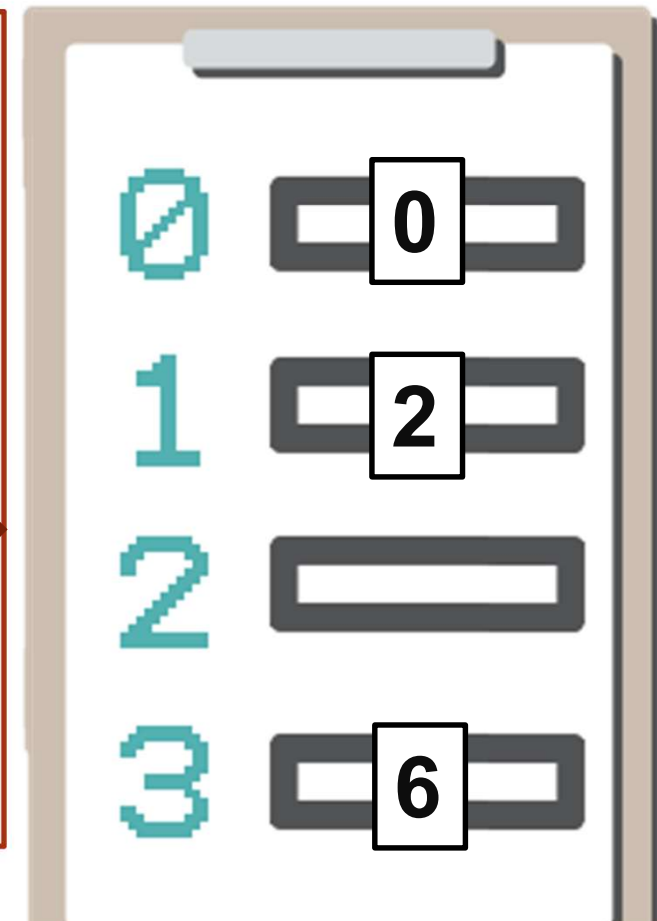


Exemplo



- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i < 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```

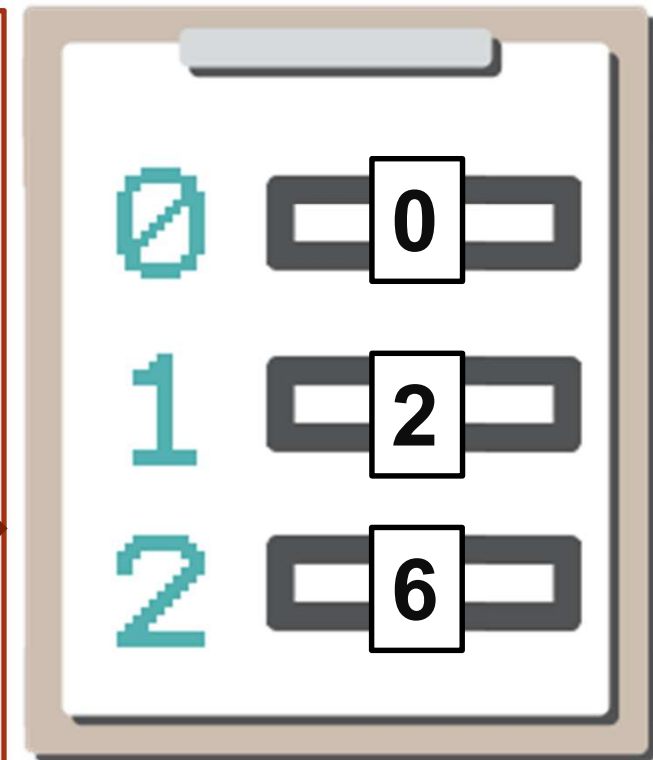


Exemplo



- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i < 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```



Exemplo



Console

0

```
List<int> minhaLista = new List<int>();

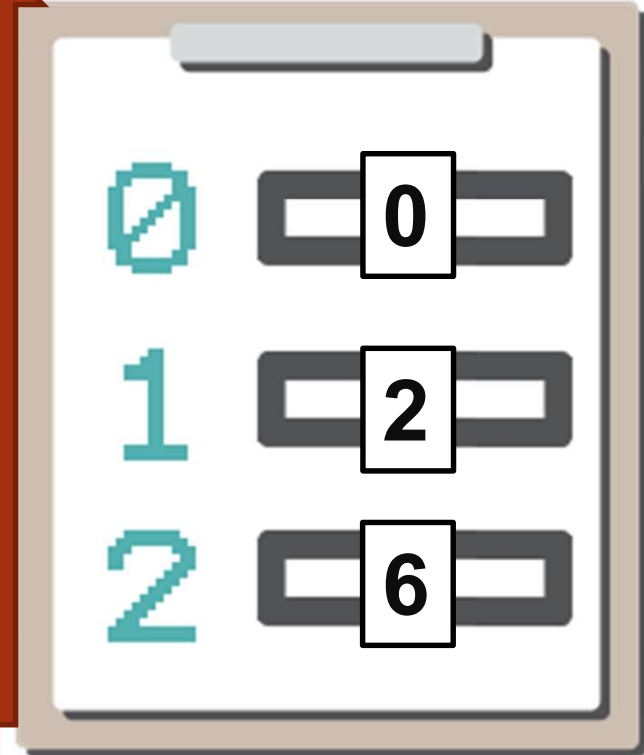
for (int i = 0; i < 3; i++)
    minhaLista.Add(i * 2);

for (int i = 0; i < minhaLista.Count; i++)
    Console.WriteLine(minhaLista[i]);

minhaLista.RemoveAt(2);

for (int i = 0; i < minhaLista.Count; i++)
    Console.WriteLine(minhaLista[i]);

Console.ReadLine();
```



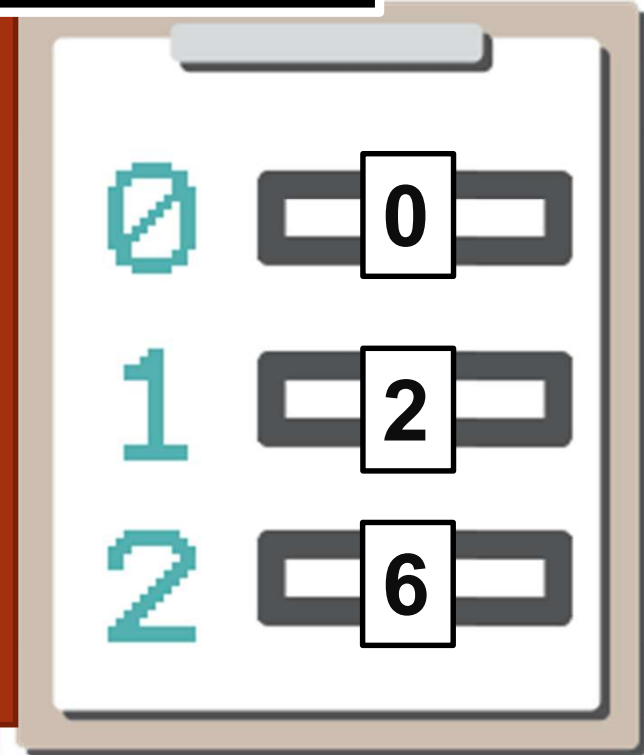
Exemplo



Console

0
2

```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i < 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```



Exemplo



Console

0

2

List<int> 6

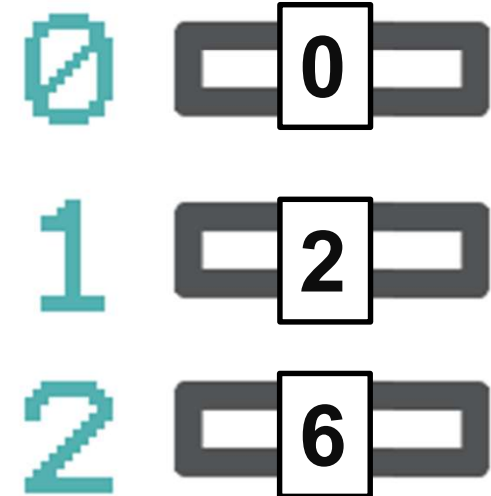
```
for (int i = 0; i < 3; i++)  
    minhaLista.Add(i * 2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
minhaLista.RemoveAt(2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
Console.ReadLine();
```



Criando uma lista

- Exemplo:

```
List<int> minhaLista;  
minhaLista = new List<int>();
```

Declara uma List que pode armazenar inteiros.

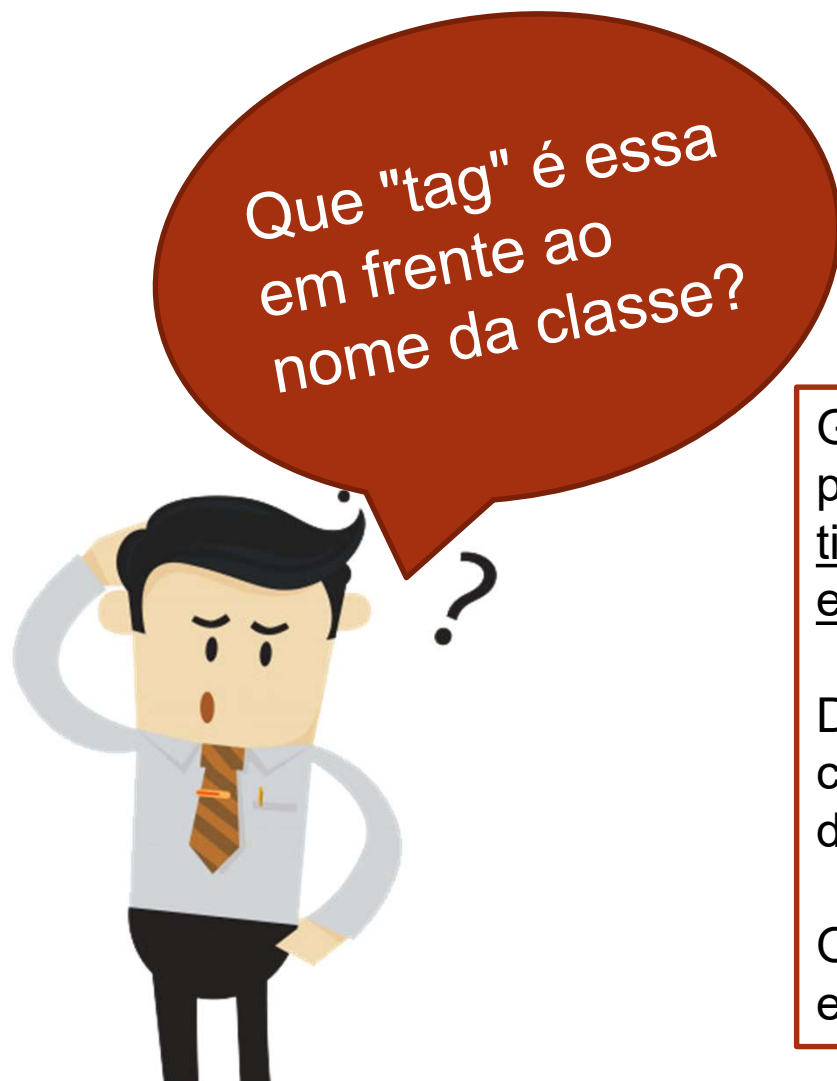
Cria um objeto List<int> chamando o seu construtor.

```
List<string> minhaLista;  
minhaLista = new List<string>();
```

Declara uma List que pode armazenar strings.

Cria um objeto List<string> chamando o seu construtor.

Criando uma lista



```
List<int> minhaLista;  
minhaLista = new List<int>();
```

Generics é um conceito criado em programação para especificar quando tipos de dados são utilizados como uma espécie de parâmetro.




Dessa forma, é possível a criação de classes que operam em dados de tipos diferentes.

O uso mais comum deste conceito é encontrado nas classes de coleção.

Listas encolhem e aumentam de tamanho dinamicamente



- Uma grande vantagem da Lista em relação a uma Array, é que você não precisa saber o tamanho dela quando você a cria.
- Ela pode aumentar e diminuir automaticamente para se adequar a necessidade do programador.
- Isso é feito através do uso dos métodos prontos e resolve os DOIS PROBLEMAS que enunciamos nas arrays!:

	Add(T)	Adiciona um elemento ao final da List<T>.
	RemoveAt(Int)	Remove o elemento no índice especificado do List<T>.
	Count	Retorna a quantidade de elementos que estão dentro da lista.

Aumentando dinamicamente



O tamanho da List não é fixo e cresce sempre que você chama o método Add()

```
Carta c1 = new Carta("Ás", "Copas");  
Carta c2 = new Carta("Dois", "Espadas");  
Carta c3 = new Carta("Valete", "Paus");  
Carta c4 = new Carta("Sete", "Ouros");  
  
List<Carta> minhasCartas = new List<Carta>();  
  
minhasCartas.Add(c1);  
minhasCartas.Add(c2);  
minhasCartas.Add(c3);  
minhasCartas.Add(c4);  
  
minhasCartas.RemoveAt(1);
```

Inicialmente lista de tamanho 0

Ao final desses quatro comandos, a lista tem tamanho 4.

Reduzindo dinamicamente



O método `RemoveAt()` não só elimina a carta da posição especificada como libera a memória da posição e rearranja a lista.

```
Carta c1 = new Carta("Ás", "Copas");  
Carta c2 = new Carta("Dois", "Espadas");  
Carta c3 = new Carta("Valete", "Paus");  
Carta c4 = new Carta("Sete", "Ouros");  
  
List<Carta> minhasCartas = new List<Carta>();  
  
minhasCartas.Add(c1);  
minhasCartas.Add(c2);  
minhasCartas.Add(c3);  
minhasCartas.Add(c4);  
  
minhasCartas.RemoveAt(1);
```

Depois deste comando a lista tem 3 elementos.

Percorrendo a lista

- A forma de se percorrer uma lista pode ser a mesma que já utilizamos com vetores.

```
List<int> minhaLista = new List<int>();
```

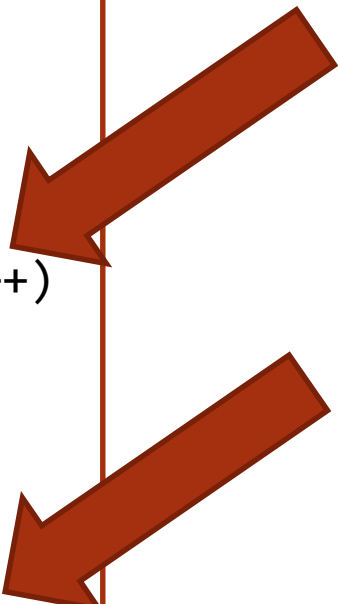
```
for (int i = 0; i < 4; i++)  
    minhaLista.Add(i * 2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
minhaLista.RemoveAt(2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
Console.ReadLine();
```



Uso do for e iterando enquanto o contador for menor que a quantidade de elementos na lista.

Percorrendo a lista

De uma maneira mais sofisticada



- C# provê um tipo especial de laço denominado **foreach** para percorrer listas (que também pode ser usado com vetores).
- Ele vai executar um comando para cada elemento na lista.
- Esse laço cria um identificador para cada elemento.

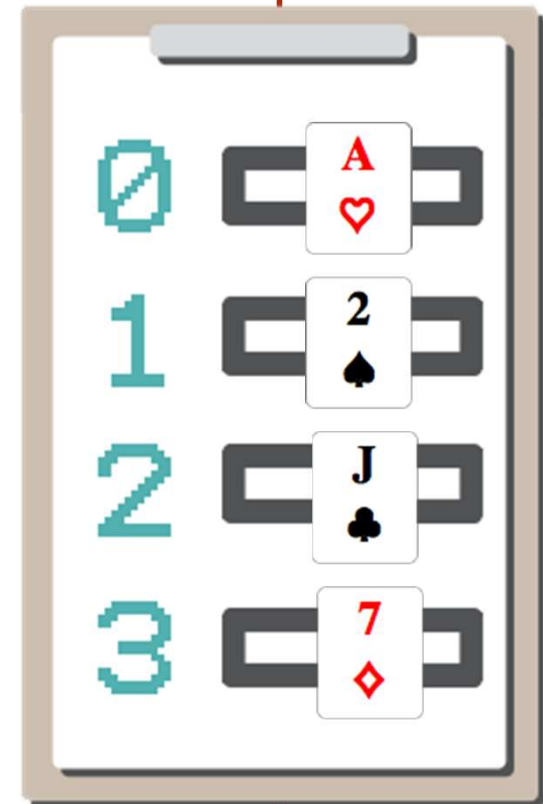
```
foreach(TipoDoDado dado in NomeDaLista)
{
    /* A cada iteração a variável dado estará se
       referindo a um elemento na lista */
}
```


Percorrendo a lista

De uma maneira mais sofisticada



```
Carta c1 = new Carta("Ás", "Copas");  
Carta c2 = new Carta("Dois", "Espadas");  
Carta c3 = new Carta("Valete", "Paus");  
Carta c4 = new Carta("Sete", "Ouros");  
  
List<Carta> minhasCartas = new List<Carta>();  
  
minhasCartas.Add(c1);  
minhasCartas.Add(c2);  
minhasCartas.Add(c3);  
minhasCartas.Add(c4);  
  
foreach(Carta umaCarta in minhasCartas)  
{  
    Console.WriteLine(umaCarta.Nome());  
}
```



Percorrendo a lista

De uma maneira mais sofisticada

Na primeira iteração do foreach

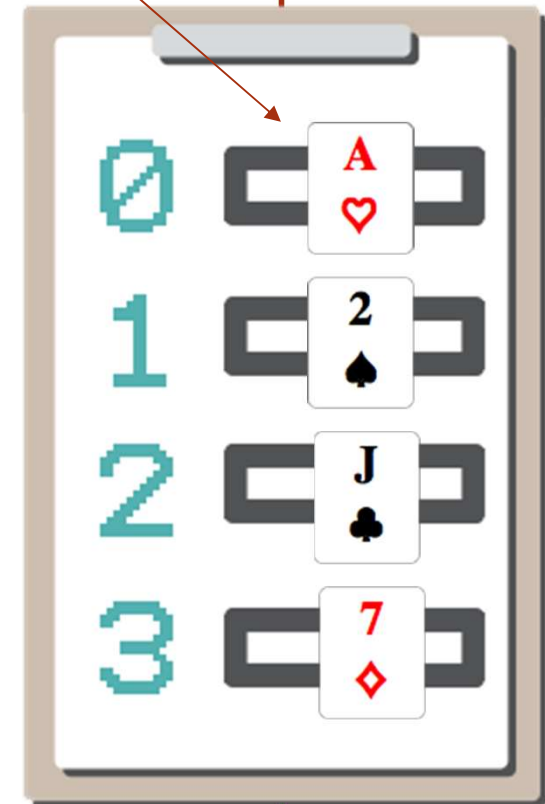
umaCarta

```
Carta c1 = new Carta("Ás", "Copas");  
Carta c2 = new Carta("Dois", "Espadas");  
Carta c3 = new Carta("Valete", "Paus");  
Carta c4 = new Carta("Sete", "Ouros");
```

```
List<Carta> minhasCartas = new List<Carta>();
```

```
minhasCartas.Add(c1);  
minhasCartas.Add(c2);  
minhasCartas.Add(c3);  
minhasCartas.Add(c4);
```

```
foreach(Carta umaCarta in minhasCartas)  
{  
    Console.WriteLine(umaCarta.Nome());  
}
```



Percorrendo a lista

De uma maneira mais sofisticada

Na segunda iteração do foreach

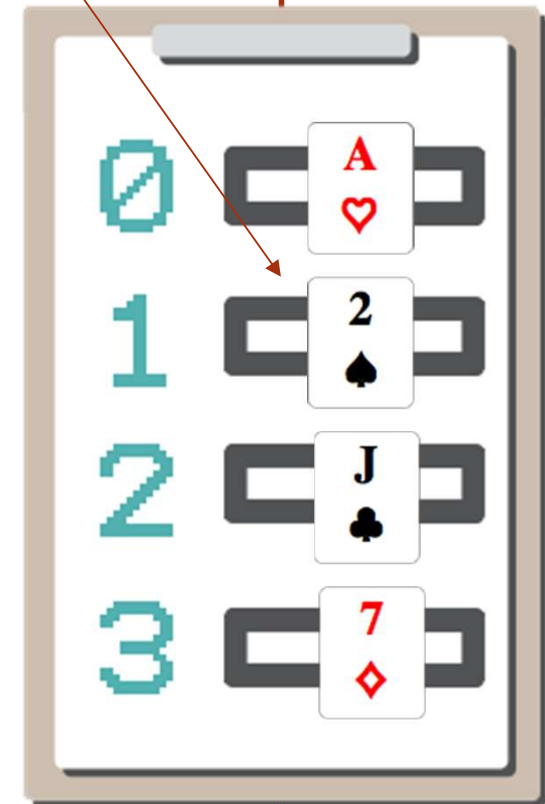
umaCarta

```
Carta c1 = new Carta("Ás", "Copas");  
Carta c2 = new Carta("Dois", "Espadas");  
Carta c3 = new Carta("Valete", "Paus");  
Carta c4 = new Carta("Sete", "Ouros");
```

```
List<Carta> minhasCartas = new List<Carta>();
```

```
minhasCartas.Add(c1);  
minhasCartas.Add(c2);  
minhasCartas.Add(c3);  
minhasCartas.Add(c4);
```

```
foreach(Carta umaCarta in minhasCartas)  
{  
    Console.WriteLine(umaCarta.Nome());  
}
```



Percorrendo a lista

De uma maneira mais sofisticada

Na terceira iteração do foreach

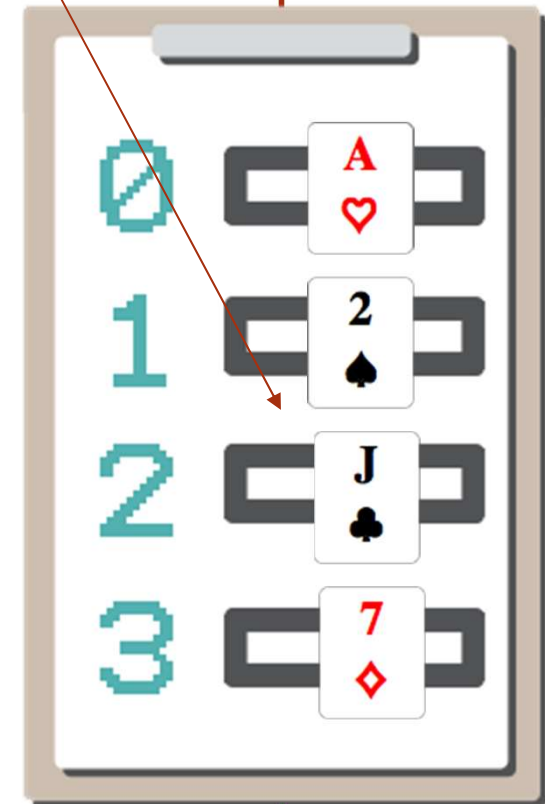
umaCarta

```
Carta c1 = new Carta("Ás", "Copas");  
Carta c2 = new Carta("Dois", "Espadas");  
Carta c3 = new Carta("Valete", "Paus");  
Carta c4 = new Carta("Sete", "Ouros");
```

```
List<Carta> minhasCartas = new List<Carta>();
```

```
minhasCartas.Add(c1);  
minhasCartas.Add(c2);  
minhasCartas.Add(c3);  
minhasCartas.Add(c4);
```

```
foreach(Carta umaCarta in minhasCartas)  
{  
    Console.WriteLine(umaCarta.Nome());  
}
```



Percorrendo a lista

De uma maneira mais sofisticada

Na quarta iteração do foreach

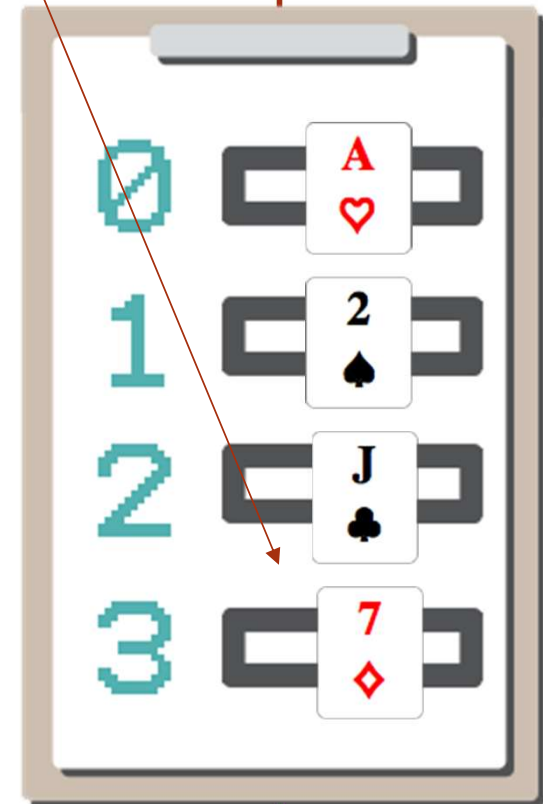
umaCarta

```
Carta c1 = new Carta("Ás", "Copas");  
Carta c2 = new Carta("Dois", "Espadas");  
Carta c3 = new Carta("Valete", "Paus");  
Carta c4 = new Carta("Sete", "Ouros");
```

```
List<Carta> minhasCartas = new List<Carta>();
```



```
minhasCartas.Add(c1);  
minhasCartas.Add(c2);  
minhasCartas.Add(c3);  
minhasCartas.Add(c4);
```

```
foreach(Carta umaCarta in minhasCartas)  
{  
    Console.WriteLine(umaCarta.Nome());  
}
```



O que mais posso fazer com listas?

- Outros Atributos/Propriedades e Métodos

Atributos/ Propriedades		Capacity	Indica o número de elementos que a lista pode suportar antes de ter que se redimensionar.
		Count	Retorna a quantidade de elementos que estão dentro da lista.
Métodos		Clear()	Remove todos os elementos da lista tornando-a uma lista vazia.
		Contains(T)	Retorna true se um determinado elemento está dentro da lista ou false caso contrário.
		IndexOf(T)	Retorna o índice de onde um determinado elemento se encontra na lista.
		Insert(Int, T)	Insere um elemento na lista na posição especificada como parâmetro.
		Sort()	Ordena os elementos da lista de acordo com um comparador padrão.

Para os curiosos...

Para saber mais sobre a classe List<T>, confira o site:
[https://msdn.microsoft.com/pt-br/library/6sh2ey19\(v=vs.110\).aspx](https://msdn.microsoft.com/pt-br/library/6sh2ey19(v=vs.110).aspx)