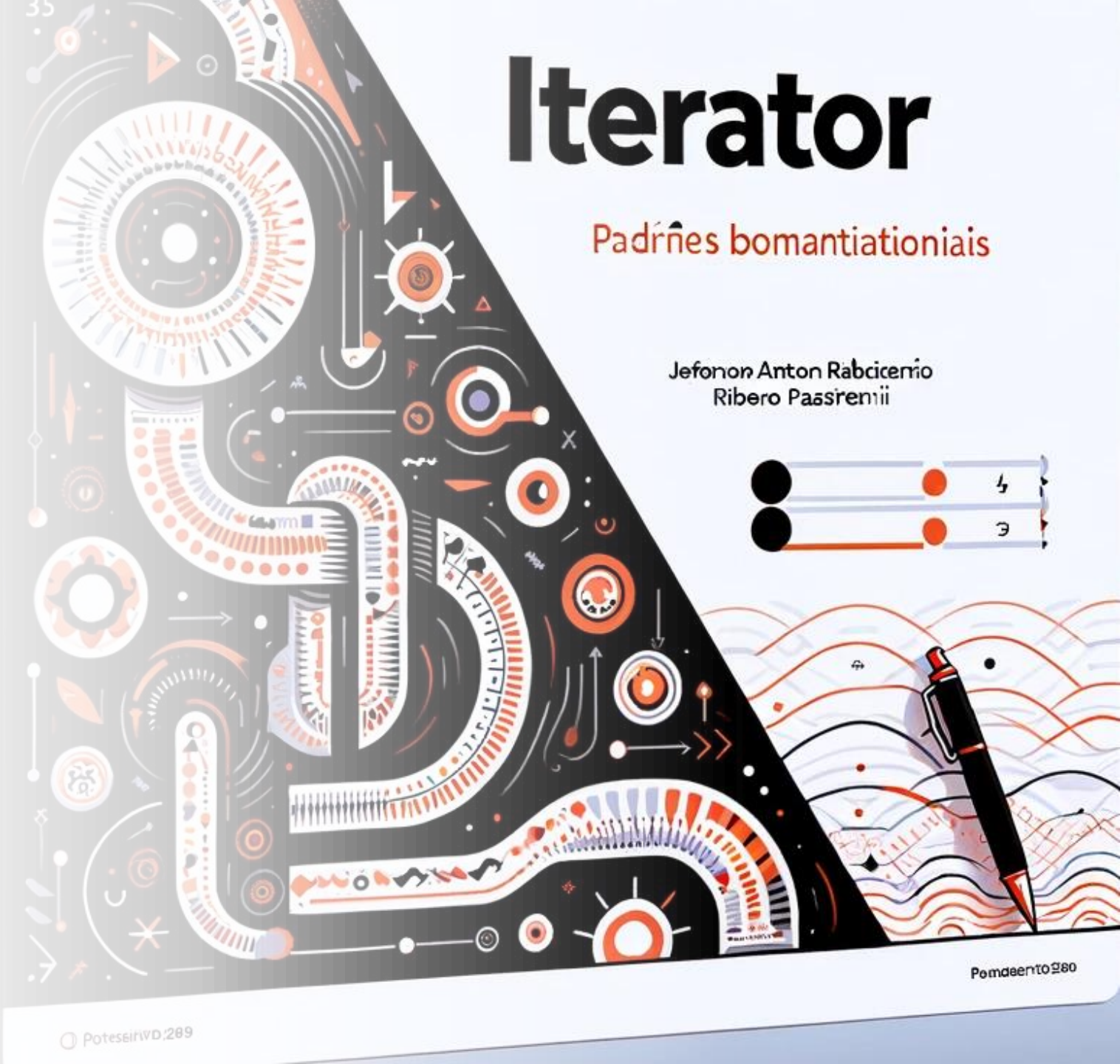
Iterator

Padrões de Projeto Comportamental II

Prof. Me Jefferson Passerini



- O padrão ***Iterator*** fornece uma maneira de acessar, sequencialmente, os elementos de um objeto agregado sem expor a sua representação subjacente.

Aplicabilidade

- Quando é necessário acessar o conteúdo de um objeto agregado sem expor sua representação interna.
- Quando é preciso suportar vários tipos de iteração em objetos agregados.
- Quando é necessário fornecer uma interface uniforme para iterar sobre diferentes estruturas agregadas (ou seja, para suportar iteração polimórfica).

Componentes

- **Cliente:** Precisa iterar sobre os objetos agregados.
- **Agregado:** Define uma interface para criação de um objeto *Iterator*. O cliente espera essa interface ao invés de agregados concretos.
- **AgregadoConcreto:** Possui uma coleção de objetos e implementa o método que retorna um objeto do tipo *Iterator* referente a sua coleção.
- **Iterator:** Fornece a interface que todos os iteradores concretos devem implementar, bem como um conjunto de métodos para acessar os elementos de um agregado.
- **IteratorConcreto:** Implementa a interface a interface de Iterator e mantém o controle da posição corrente no percurso do agregado.

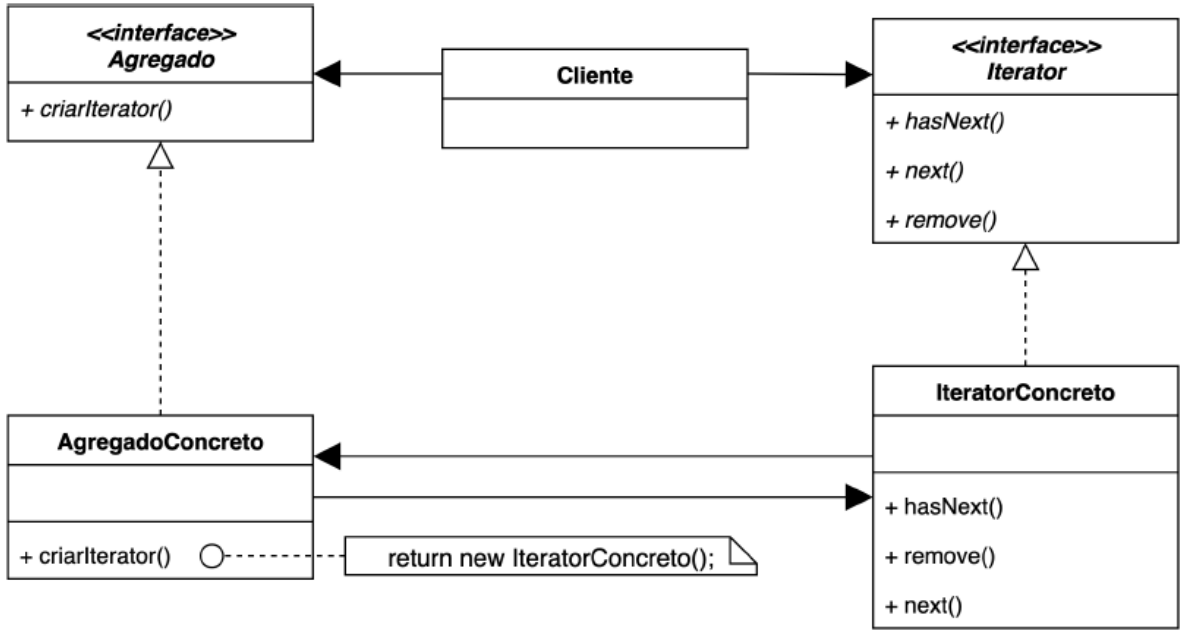


Diagrama de Classes

Motivação (Porque utilizar?)

- Um grupo de objetos pode ser chamado de coleção ou agregados.
- Tais objetos podem estar armazenados em diferentes estruturas de dados, como listas, matrizes, pilhas, árvores entre outras.
- É interessante que forneçam uma maneira de acessar seus elementos sem expor sua estrutura de dados interna.
- O padrão de projeto *Iterator* torna isso possível. A ideia principal deste padrão é assumir a responsabilidade pela iteração de ponta a ponta sobre os elementos de objetos agregados.
- Isso é feito por meio de uma classe *Iterator*. Tal classe implementa uma interface que dita quais métodos são necessários para acessar os elementos de diferentes objetos agregados.

Matriz ($matriz)

	0	1	2
0	Objeto 1	Objeto 2	Objeto 3
1	Objeto 4	Objeto 5	Objeto 6
2	Objeto 7	Objeto 8	Objeto 9

Exemplo de agregados

Lista ($lista)

Objeto 1	Objeto 2	Objeto 3	Objeto 4	Objeto 5	Objeto 6	Objeto 7	Objeto 8	Objeto 9
0	1	2	3	4	5	6	7	8

Motivação (Porque utilizar?)

- Os código ao lado são simples, pois trata-se de estruturas de dados simples, mas para percorrer suas estruturas internas devemos conhece-las para cada uma das estruturas.

- Como poderíamos, com apenas um código, iterar sobre a lista e a matriz, e além disso ainda ocultar suas estruturas internas para o cliente?

Matriz ($matriz)

	0	1	2
0	Objeto 1	Objeto 2	Objeto 3
1	Objeto 4	Objeto 5	Objeto 6
2	Objeto 7	Objeto 8	Objeto 9

Exemplo de agregados

```php
$qtdLinhas = 3; //Quantidade de linhas da matriz.
$qtdColunas = 3; //Quantidade de colunas da matriz.

//Para cada linha da matriz.
for ($linha = 0; $linha < $qtdLinhas; $linha++) {
    //Para cada coluna da matriz.
    for ($coluna = 0; $coluna < $qtdColunas; $coluna++) {
        echo $matriz[$linha][$coluna] . ' '; //Imprima o item;
    }
}
```

```php
$tamanho = 9; //Tamanho da lista

//Para cada elemento da lista
for ($i = 0; $i < $tamanho; $i++) {
    echo $lista[$i] . ' '; //Imprima o item;
}
```

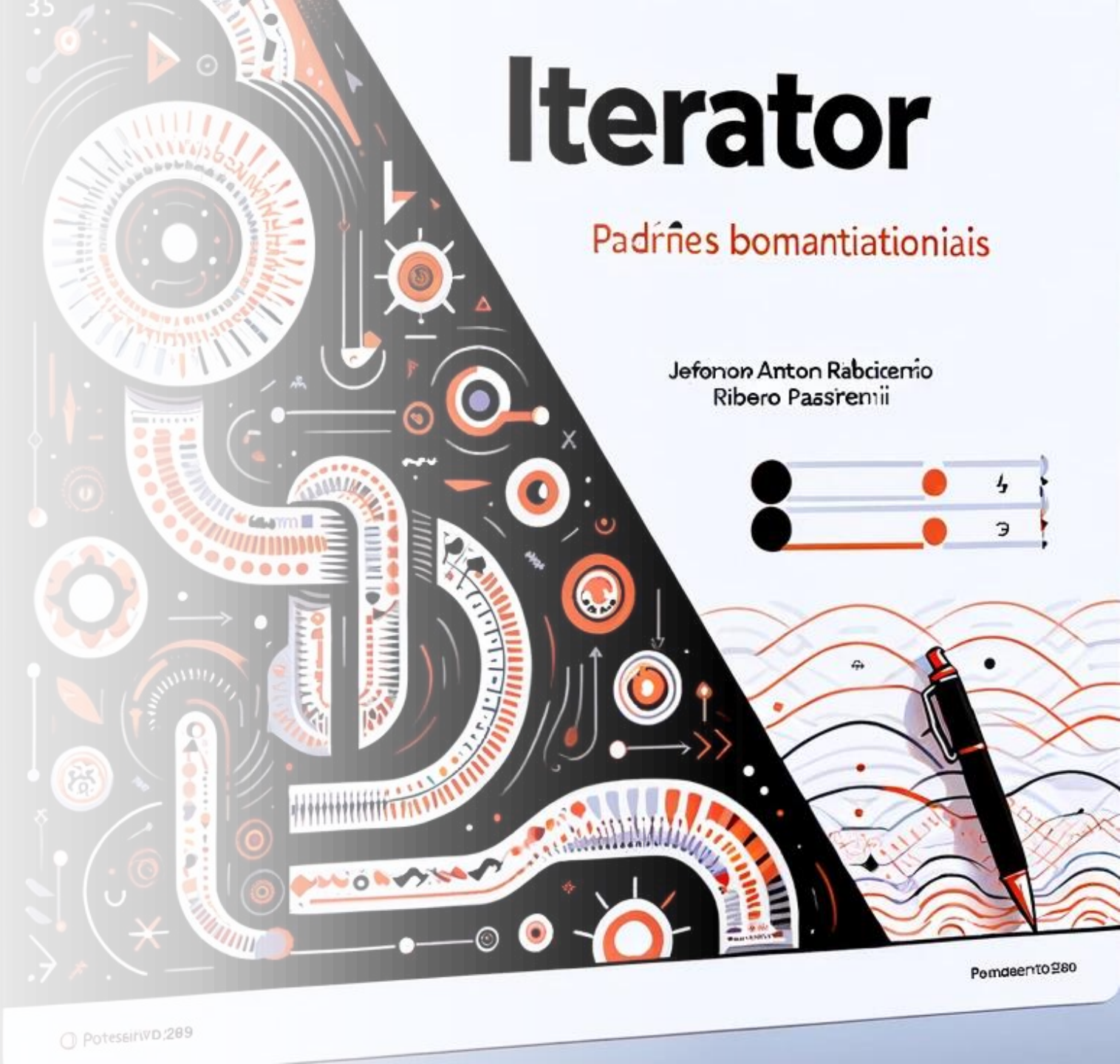
Lista ($lista)

Objeto 1	Objeto 2	Objeto 3	Objeto 4	Objeto 5	Objeto 6	Objeto 7	Objeto 8	Objeto 9
0	1	2	3	4	5	6	7	8

Iterator – Implementação C#

Padrões de Projeto Comportamental II

Prof. Me Jefferson Passerini



Iterator – C# - Exemplo Inicial

- O padrão *Iterator* tem como objetivo unificar e encapsular a iteração em agregados.

- Isso é possível por meio de uma interface comum implementada por classes do tipo *Iterator*, que por sua vez, encapsulam a forma de iteração de cada tipo de objeto agregado, expondo métodos padronizados independente da estrutura de dados do objeto agregado sobre qual atuam

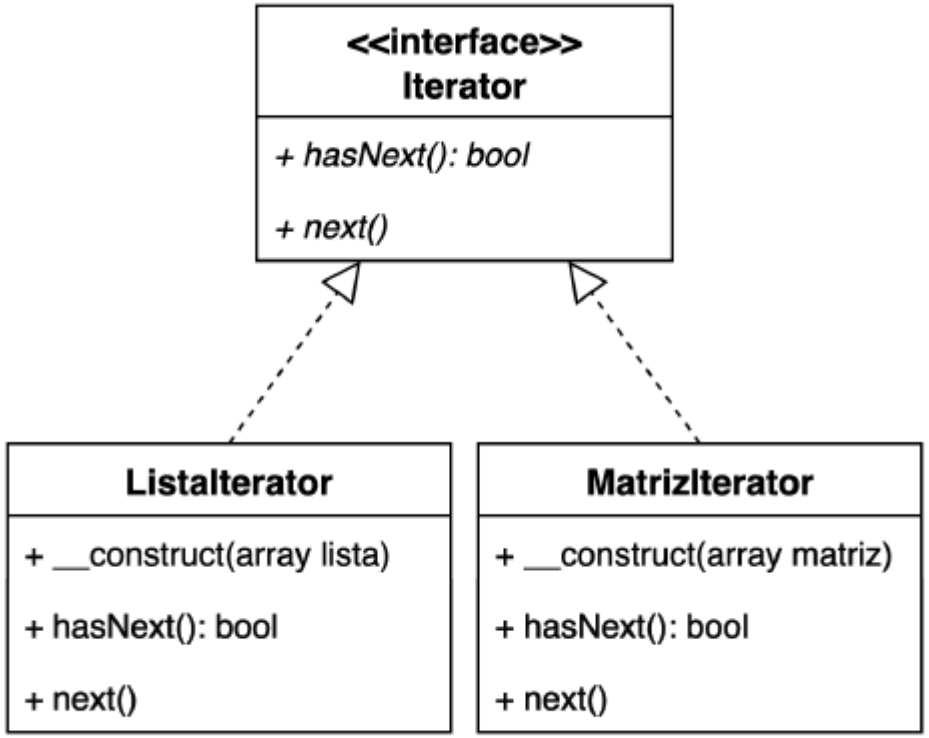


Diagrama de classes dos Iterators

Iterator – C# - Exemplo Inicial

- A interface *iterator* exige que uma classe implemente dois métodos.

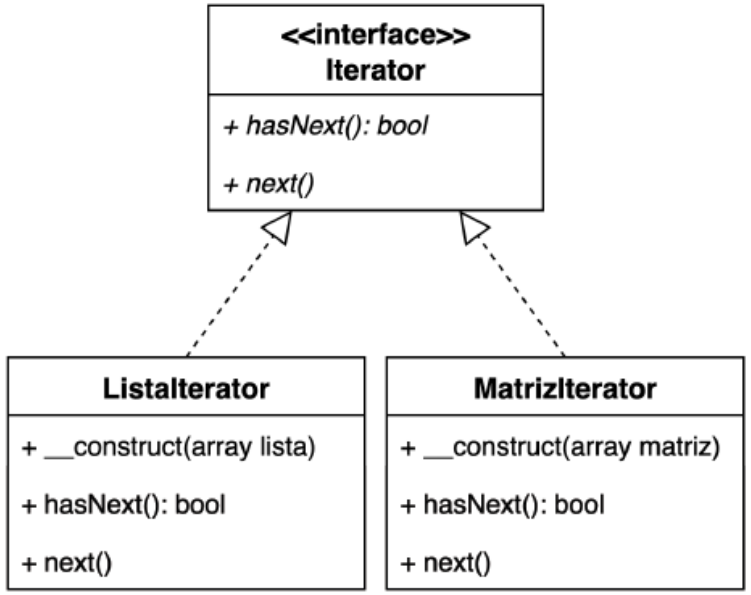
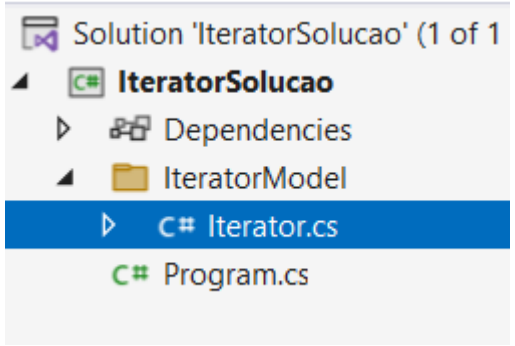


Diagrama de classes dos Iterators

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace IteratorSolucao.IteratorModel
8   {
        0 references
9       public interface Iterator
10      {
            0 references
11          public Boolean hasNext();
            0 references
12          public void next();
13      }
14  }
```



Iterator – C# - Exemplo Inicial

- Vamos criar uma classe que gerencie uma **lista** utilizando o padrão Iterator.

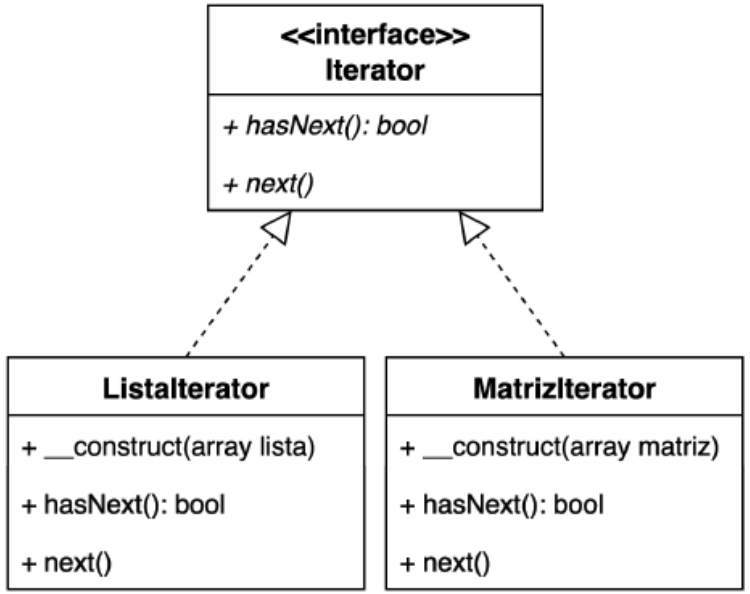
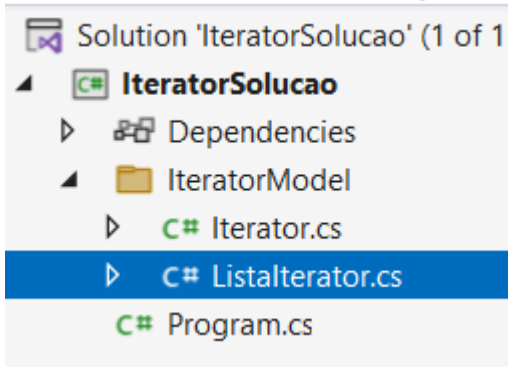


Diagrama de classes dos Iterators




```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace IteratorSolucao.IteratorModel
9  {
       2 references
10     public class ListaIterator : Iterator
11     {
12         private List<int> lista = new List<int>();
           3 references
13         private int indice { get; set; } = 0;
           2 references
14         private int tamanho { get; set; }
15
           1 reference
16         public ListaIterator(List<int> lista) {
17             this.lista = lista;
18             this.tamanho = lista.Count();
19         }
20
           2 references
21         public bool hasNext() {
22             if (this.indice >= this.tamanho)
23             {
24                 return false;
25             }
26             return true;
27         }
28
29         public int next() {
30            int item = this.lista[this.indice];
31             this.indice += 1;
32             return item;
33         }
34     }
35 }
```

Iterator – C# - Exemplo Inicial

- Vamos criar uma classe que gerencie uma **matriz** utilizando o

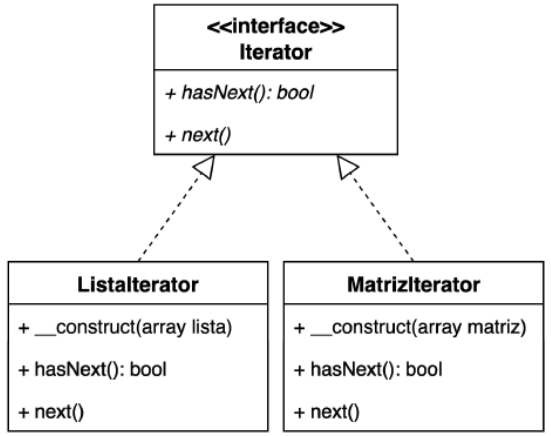
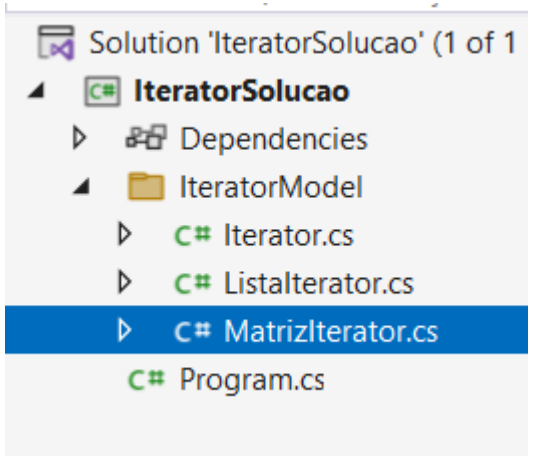


Diagrama de classes dos Iterators

```php
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IteratorSolucao.IteratorModel
{
    2 references
    public class MatrizIterator : Iterator
    {
        public List<List<int>> matriz = new List<List<int>>();
        4 references
        public int indiceLinha { get; set; } = 0;
        4 references
        public int indiceColuna { get; set; } = 0;
        2 references
        public int qtdLinhas { get; set; }
        2 references
        public int qtdColunas { get; set; }

        1 reference
        public MatrizIterator(List<List<int>> matriz)
        {
            this.matriz = matriz;
            this.qtdLinhas = matriz.Count;
            this.qtdColunas = matriz.Count > 0 ? matriz[0].Count : 0;
        }
```




```php
        public void incrementarIndice()
        {
            if (this.indiceColuna >= this.qtdColunas-1)
            {
                this.indiceLinha = this.indiceLinha + 1;
                this.indiceColuna = 0;
            } else
            {
                this.indiceColuna += 1;
            }
        }

        2 references
        public bool hasNext()
        {
            if(this.indiceLinha > (this.qtdLinhas - 1))
            {
                return false;
            }
            return true;
        }

        2 references
        public int next()
        {
            int item = this.matriz[this.indiceLinha][this.indiceColuna];
            this.incrementarIndice();
            return item;
        }
    }
}
```

Iterator – C# - Exemplo Inicial

- toda a lógica de iteração sobre os itens ficou sob a responsabilidade de cada classe. O cliente pode utilizar um único código para iterar sobre os elementos tanto da **$lista** quanto da **$matriz**.
- Vamos criar uma pequena classe que itera sobre os elementos de um *Iterator*.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace IteratorSolucao.IteratorModel
8   {
        0 references
9       public class ImpressoraDeAgregados
10      {
            0 references
11          public static void iterar(Iterator iterator)
12          {
13              while (iterator.hasNext())
14              {
15                  Console.WriteLine(iterator.next() + "  ");
16              }
17          }
18      }
19  }
```

Iterator – C# - Exemplo Inicial

- Vamos realizar o teste.

```
1   //cria lista e matriz
2   using IteratorSolucao.IteratorModel;
3
4   List<int> lista = [1, 2, 3, 4, 5, 6, 7, 8, 9];
5   List<List<int>> matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
6
7   //criar o iterator
8   Iterator listaIterator = new ListaIterator(lista);
9   Iterator matrizIterator = new MatrizIterator(matriz);
10
11  //iterar sobre a lista
12  Console.WriteLine("Lista - vetor");
13  ImpressoraDeAgregados.iterar(listaIterator);
14
15  //iterar sobre a matriz
16  Console.WriteLine("Matriz");
17  ImpressoraDeAgregados.iterar(matrizIterator);
18
```

```
Lista - vetor
1
2
3
4
5
6
7
8
9
Matriz
1
2
3
4
5
6
7
8
9
```

Iterator C# - Exemplo Completo

- Vamos criar os agregados propriamente ditos. Poderíamos implementar nos agregados métodos para diversas finalidades, tais como:
  - Adicionar um item;
  - Recuperar um item em uma posição específica;
  - Remover um item;
  - Atualizar um item;
  - Recuperar tamanho do agregado.

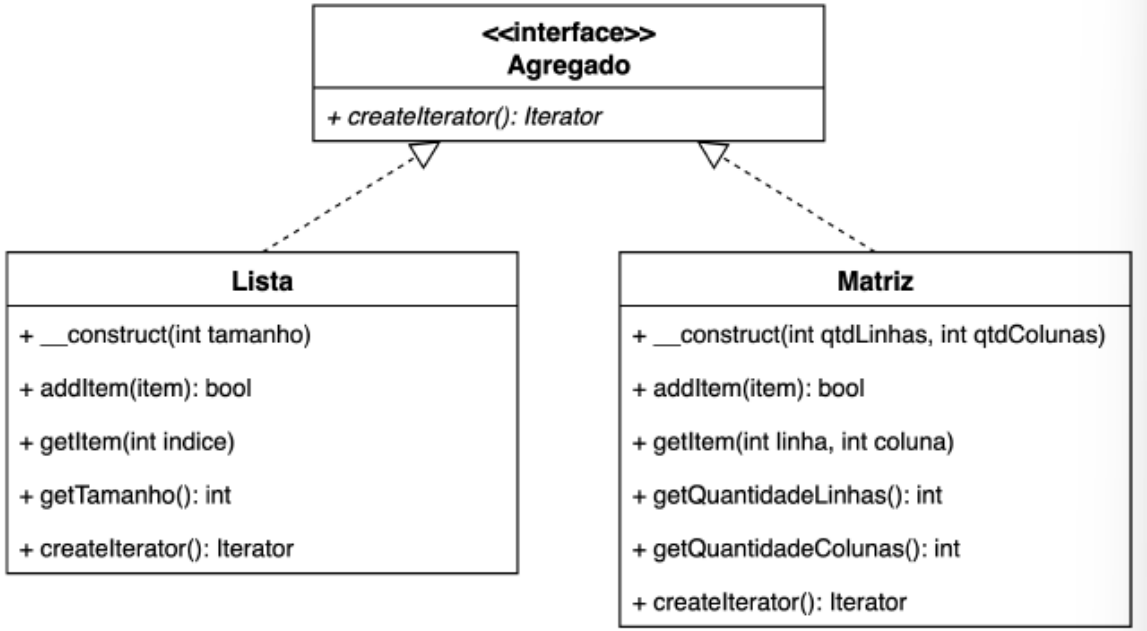


Diagrama de classe dos agregados

- Porém nosso objetivo é entender o padrão *Iterator*, por isso vamos manter o código simples e implementar somente o necessário, serão eles os métodos para adicionar e recuperar itens e também métodos para recuperar o tamanho do agregado.

Iterator C# - Exemplo Completo

- Interface Agregado.
- Os agregados iram implementar a mesma interface

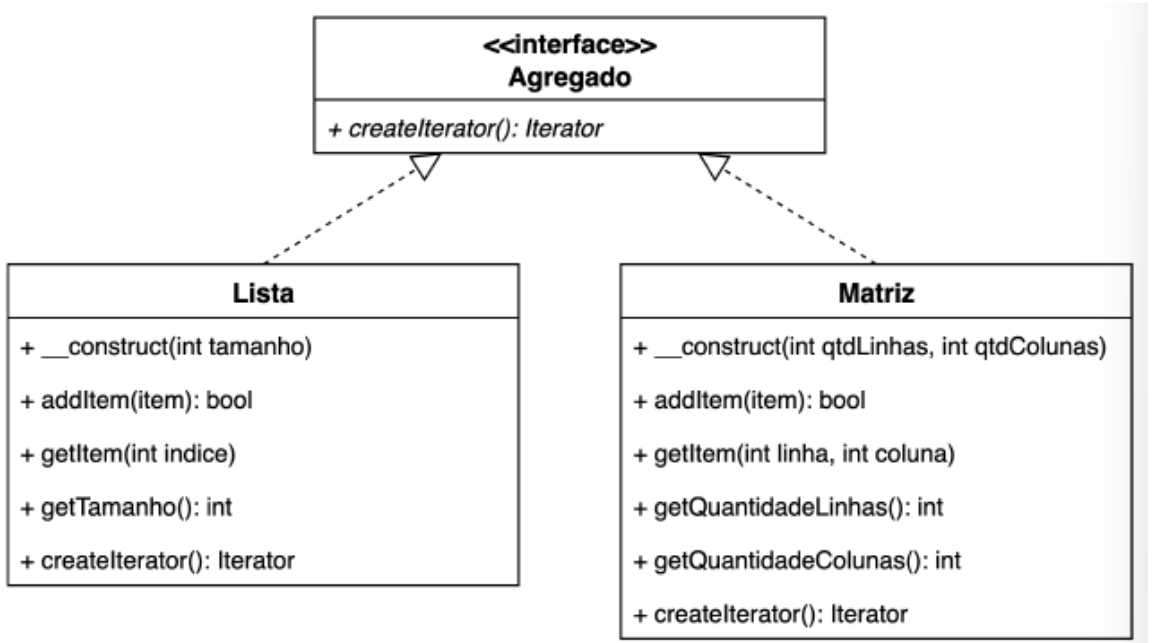


Diagrama de classe dos agregados

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace IteratorSolucao.IteratorModel
8   {
        3 references
9       public interface Agregado
10      {
            3 references
11          public Iterator createIterator();
12      }
13  }
```

Iterator C# - Exemplo Completo

- Classe Lista.

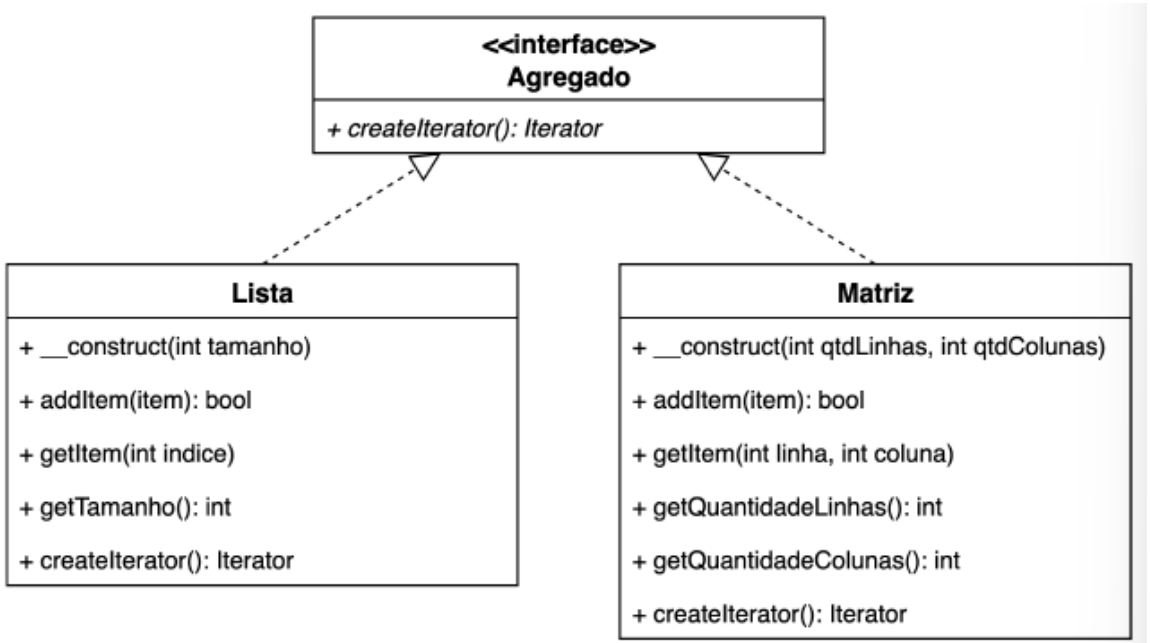


Diagrama de classe dos agregados

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IteratorSolucao.IteratorModel {
    5 references
    public class Lista : Agregado{

        private List<int> lista = new List<int>();
        private int tamanho;

        1 reference
        public Lista(int tamanho)
        {
            this.tamanho = tamanho;
        }

        9 references
        public Boolean addItem(int item)
        {
            if (this.lista.Count() <= (this.tamanho - 1))
            {
                this.lista.Add(item);
            }
            return true;
        }

        1 reference
        public int getTamanho()
        {
            return this.tamanho;
        }

        1 reference
        public int getItem(int indice)
        {
            return this.lista[indice];
        }

        2 references
        public Iterator createIterator()
        {
            return new ListaIterator(this);
        }
    }
}
```

Iterator C# - Exemplo Completo

- Classe Matriz.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IteratorSolucao.IteratorModel
{
    5 references
    public class Matriz : Agregado
    {
        private int[,] matriz;
        3 references
        public int quatidadeLinhas { get; set; }
        4 references
        public int quantidadeColunas { get; set; }
        3 references
        public int linhaAtual { get; set; }
        5 references
        public int colunaAtual { get; set; }

        1 reference
        public Matriz(int qtdLinhas, int qtdColunas)
        {
            matriz = new int[qtdLinhas,qtdColunas];
            this.quatidadeLinhas = qtdLinhas;
            this.quantidadeColunas = qtdColunas;
        }

        1 reference
        public int getItem(int linha, int coluna)
        {
            return this.matriz[linha,coluna];
        }

        public int getQuantidadeLinhas()
        {
            return this.quatidadeLinhas;
        }

        public Boolean addItem(int item)
        {
            //se chegou a ultima linha e coluna
            if (this.linhaAtual == (this.quatidadeLinhas) &&
                this.colunaAtual == (this.quantidadeColunas))
            {
                return false;
            }
            //se chegou na ultima coluna
            if (this.colunaAtual == (this.quantidadeColunas))
            {
                this.linhaAtual += 1;
                this.colunaAtual = 0;
            }
            //insere item
            this.matriz[this.linhaAtual, this.colunaAtual] = item;
            this.colunaAtual += 1;
            return true;
        }

        1 reference
        public int getQuantidadeColunas()
        {
            return this.quantidadeColunas;
        }

        2 references
        public Iterator createIterator()
        {
            return new MatrizIterator(this);
        }
    }
}
```

Iterator C# - Exemplo Completo

```
1   using System;
2   using System.Collections;
3   using System.Collections.Generic;
4   using System.Linq;
5   using System.Text;
6   using System.Threading.Tasks;
7
8   namespace IteratorSolucao.IteratorModel {
        2 references
9       public class ListaIterator : Iterator
10      {
11          private Lista lista;
            3 references
12          private int indice { get; set; } = 0;
            2 references
13          private int tamanho { get; set; }
14
            1 reference
15          public ListaIterator(Lista lista) {
16              this.lista = lista;
17              this.tamanho = lista.getTamanho();
18          }
19
            2 references
20          public bool hasNext() {
21              if (this.indice >= this.tamanho)
22              {
23                  return false;
24              }
25              return true;
26          }
27
28          public int next() {
29              int item = this.lista.getItem(this.indice);
30              this.indice += 1;
31              return item;
32          }
33      }
34  }
```

- Agora precisamos adaptar os *iterators* **ListaIterator** e **MatrizIterator** para atuar sobre os agregados que acabamos de criar.
- Altere seu código de acordo com o exemplo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IteratorSolucao.IteratorModel {
    2 references
    public class MatrizIterator : Iterator
    {
        public Matriz matriz;
        4 references
        public int indiceLinha { get; set; } = 0;
        4 references
        public int indiceColuna { get; set; } = 0;
        2 references
        public int qtdLinhas { get; set; }
        2 references
        public int qtdColunas { get; set; }

        1 reference
        public MatrizIterator(Matriz matriz)
        {
            this.matriz = matriz;
            this.qtdLinhas = matriz.getQuantidadeLinhas();
            this.qtdColunas = matriz.getQuantidadeColunas();
        }

        public void incrementarIndice()
        {
            if (this.indiceColuna >= this.qtdColunas-1)
            {
                this.indiceLinha = this.indiceLinha + 1;
                this.indiceColuna = 0;
            } else
            {
                this.indiceColuna += 1;
            }
        }

        2 references
        public bool hasNext()
        {
            if(this.indiceLinha > (this.qtdLinhas - 1))
            {
                return false;
            }
            return true;
        }

        2 references
        public int next()
        {
            int item = this.matriz.getItem(this.indiceLinha,this.indiceColuna);
            this.incrementarIndice();
            return item;
        }
    }
}
```

Iterator C# - Exemplo Completo

- Agora precisamos adaptar os *iterators* `ListaIterator` e `MatrizIterator` para atuar sobre os agregados que acabamos de criar.
- Altere seu código de acordo com o exemplo.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Runtime.CompilerServices;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace IteratorSolucao.IteratorModel
9  {
       2 references
10     public class ImpressoraDeAgregados
11     {
           3 references
12         public Agregado agregado { get; set; }
13
           2 references
14         public void iterar()
15         {
16             Iterator iterator = this.agregado.createIterator();
17             while (iterator.hasNext())
18             {
19                 Console.WriteLine(iterator.next() + "  ");
20             }
21         }
22     }
23 }
```

Iterator C# - Exemplo Completo

- Implemente a classe **ImpressoraDeAgregados**

Iterator C# - Exemplo Completo

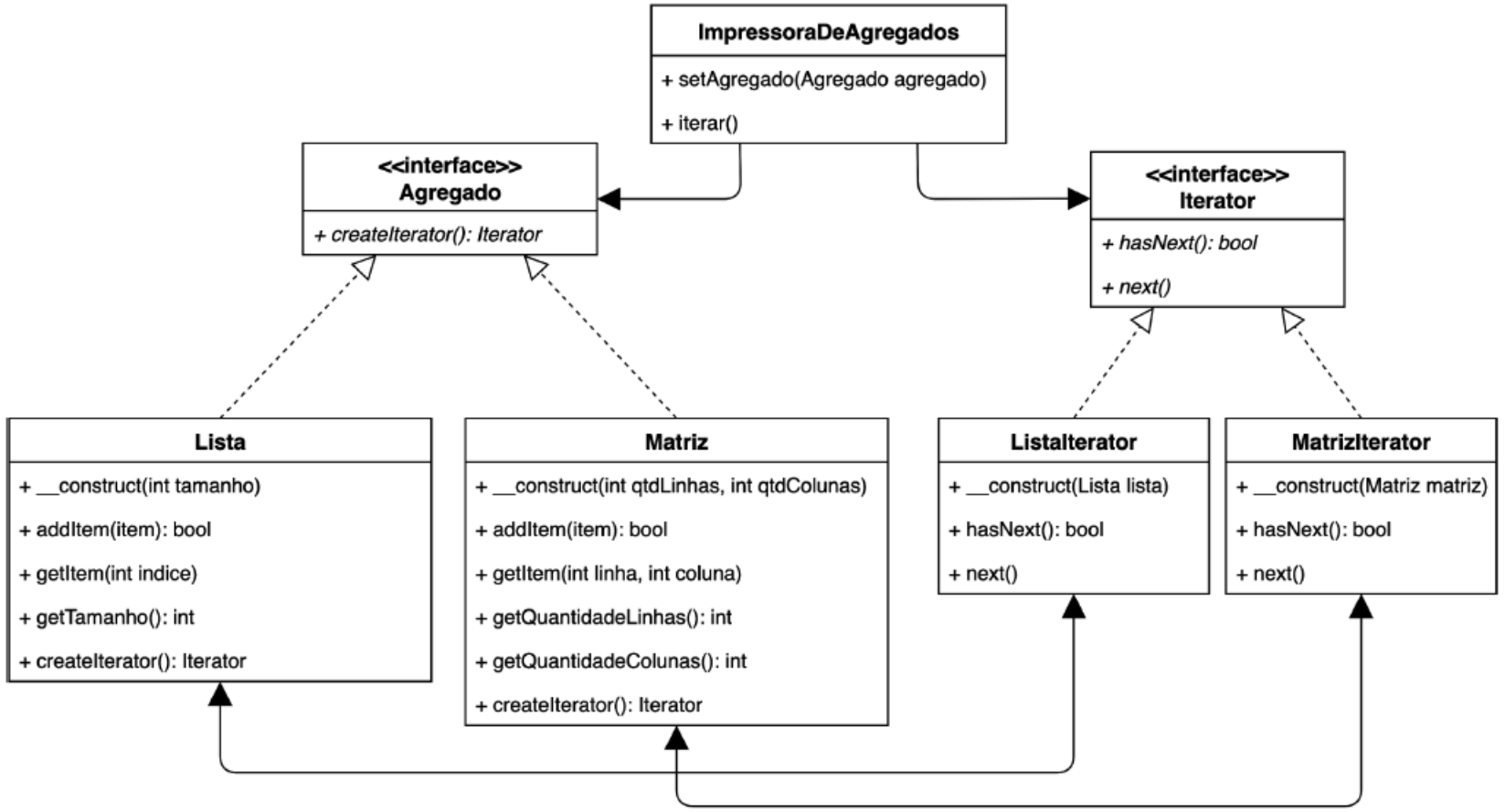
- Altere a Program (código cliente).
- Execute o teste.

```
1   //cria lista e matriz
2   using IteratorSolucao.IteratorModel;
3   using System.Net.Http.Json;
4
5   Lista lista = new Lista(9);
6   lista.addItem(1);
7   lista.addItem(2);
8   lista.addItem(3);
9   lista.addItem(4);
10  lista.addItem(5);
11  lista.addItem(6);
12  lista.addItem(7);
13  lista.addItem(8);
14  lista.addItem(9);
15
16  Matriz matriz = new Matriz(3, 3);
17  matriz.addItem(1);
18  matriz.addItem(2);
19  matriz.addItem(3);
20  matriz.addItem(4);
21  matriz.addItem(5);
22  matriz.addItem(6);
23  matriz.addItem(7);
24  matriz.addItem(8);
25  matriz.addItem(9);
26
27  ImpressoraDeAgregados cliente = new ImpressoraDeAgregados();
28
29  //iterar sobre a lista
30  cliente.agregado = lista;
31  Console.WriteLine("Lista - vetor");
32  cliente.iterar();
33
34  //iterar sobre a matriz
35  cliente.agregado = matriz;
36  Console.WriteLine("Matriz");
37  cliente.iterar();
38
39
```

```
Lista - vetor
1
2
3
4
5
6
7
8
9
Matriz
1
2
3
4
5
6
7
8
9
```

Iterator C# - Exemplo Completo

- Toda a responsabilidade de como iterar sobre os elementos são das classes que implementam o **Iterator** que são **ListaIterator** e **MatrizIterator;**
- Essa responsabilidade deixa de ser do cliente. Da mesma forma a criação do ***Iterator*** e a adição de elementos passam a ser responsabilidade das classe **Lista** e **Matriz** que implementam a interface **Agregado.**
- Com o ***Iterator*** é possível implementar novos tipos de agregados, de modo que ao passá-los para o cliente, nada precise ser modificado nele.

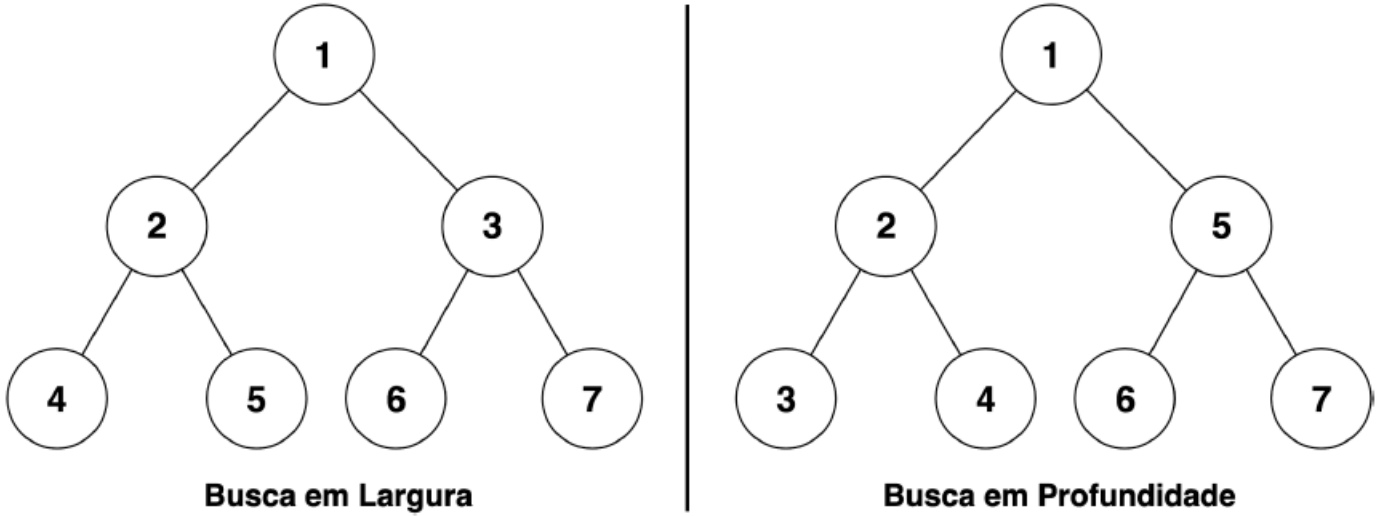


Cada agregado precisa retornar um *Iterador* Concreto e cada *Iterator* concreto gerência um agregado.

Diagrama de classe dos agregados

Iterator C# - Exemplo Completo

- No nosso exemplo foram utilizadas estruturas de dados e iterações simples, porém o iterator também pode ser utilizado para iterar sobre estruturas de dados mais complexas, como um grafo por exemplo, fazendo uma busca em largura ou busca em profundidade.

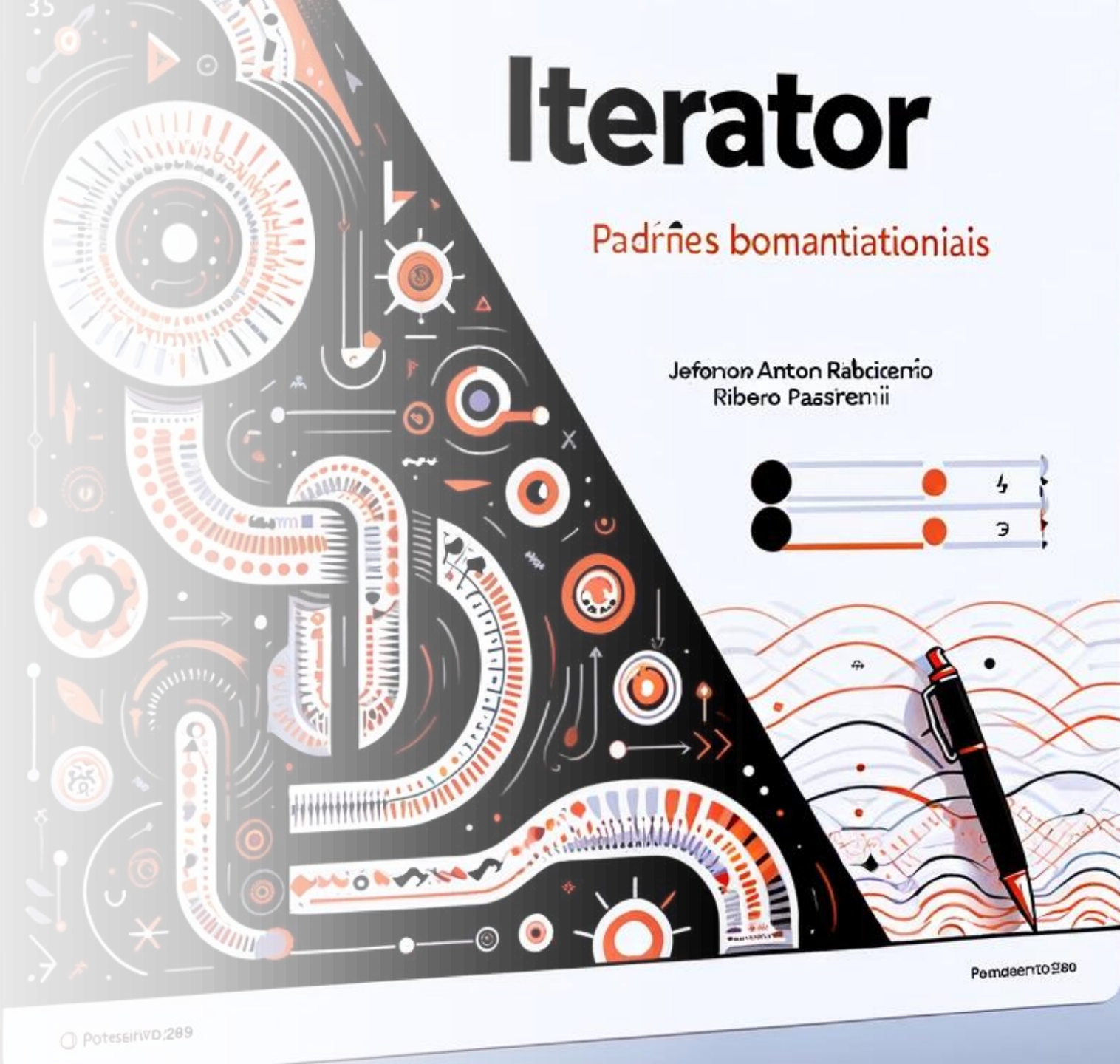


Exemplo de dois tipos de interações sobre grafos

Iterator – Implementação Java

Padrões de Projeto Comportamental II

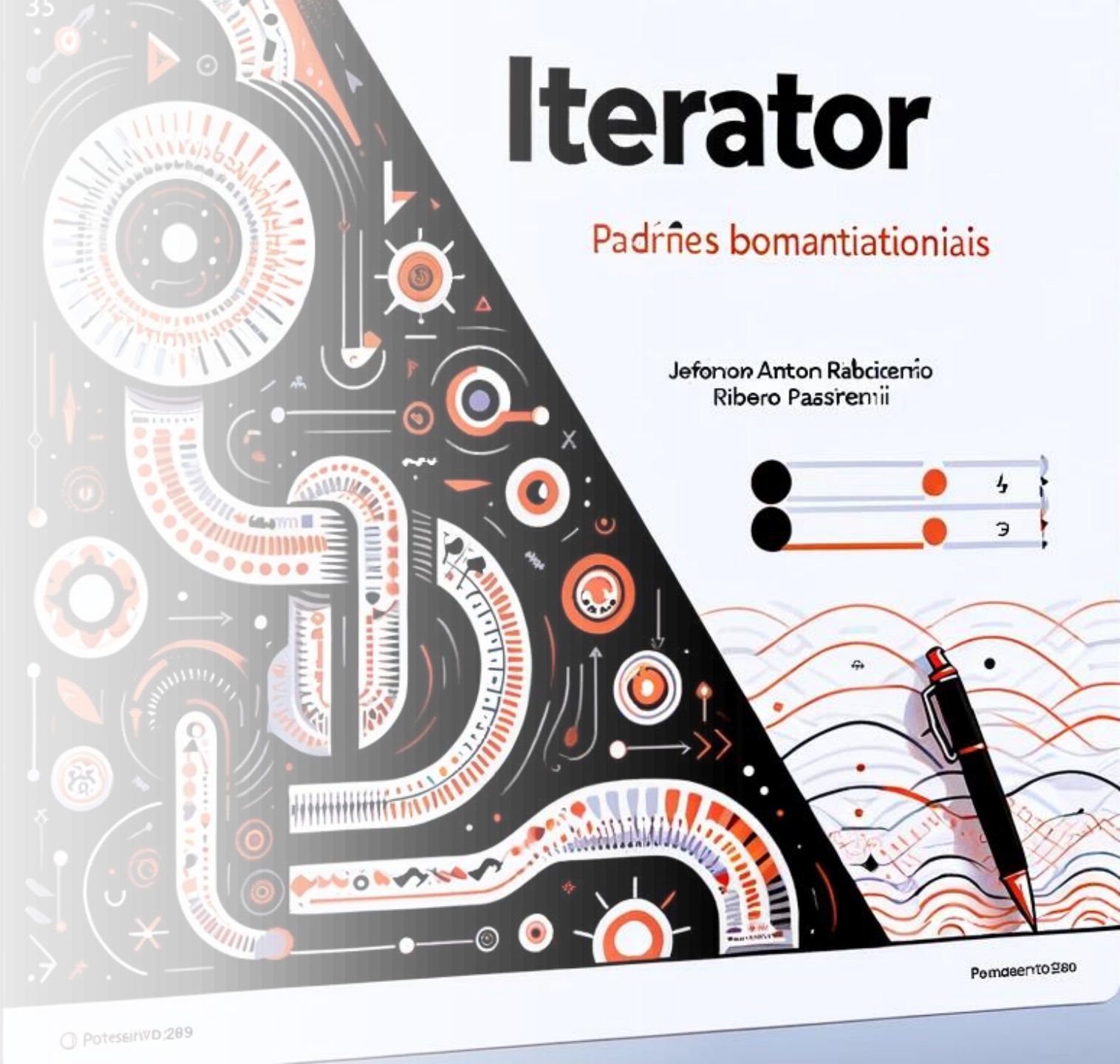
Prof. Me Jefferson Passerini



Iterator – Implementação Consequências

Padrões de Projeto Comportamental II

Prof. Me Jefferson Passerini



Iterator - Consequências

- **Suporte a variações no percurso de um agregado:** Agregados complexos podem ser percorridos de várias maneiras, encapsulando e simplificando sua iteração para objetos externos (Clientes).
- **Simplificação da interface dos agregados:** Já que os *Iterators* são os responsáveis pelas iterações, os agregados não precisam implementar métodos para tal finalidade.
- **Padronização da forma como é feita a iteração em objetos agregados de diferentes tipos (iteração polimórfica).**
- **Um agregado pode sofrer mais de uma iteração ao mesmo tempo:** Um *iterator* controla seu próprio estado de iteração. Portanto, podem existir mais de uma iteração sendo executada sobre o mesmo objeto agregado ao mesmo tempo.
- **O código passa a seguir o *"princípio de responsabilidade única"*.** Toda a responsabilidade de como iterar sobre os elementos é da classe que implementa o *Iterator.* Essa responsabilidade deixa de ser do cliente e do agregado.
- **O código também passa a seguir o *"Princípio Aberto/Fechado"*.** É possível implementar novos tipos de agregados/coleções seguindo o padrão *Iterator*, ao passá-los para o cliente existente nada precisará ser modificado nele.