UNIVERSIDADE DE PERNAMBUCO
CAMPUS GARANHUNS

Iterator

Disciplina: Padrões de Projeto
Prof.: Ewerton Mendonça
Aluno: Lucas Cassiano César
2025.2

O que veremos

- Definição;
- Características;
- Motivação;
- Onde utilizar;
- Diagrama UML;
- Participantes;
- Demonstração:
  - Domínio do problema;
  - O que varia e o que não varia;
  - Diagrama da solução;
  - Análise dos resultados;
  - Demonstração.

Definição

O iterator é um padrão que fornece um meio de acesso sequencial aos elementos de um objeto agregado sem expor a sua representação subjacente.

Características

- É um Padrão Comportamental;
- Princípio da Responsabilidade Única;
- Princípio Aberto/Fechado

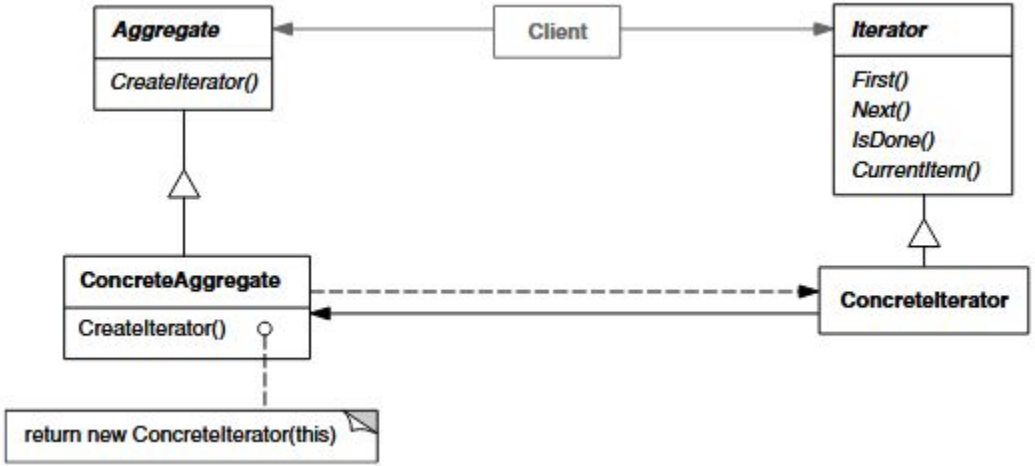
Motivação

A motivação para o uso do Iterator surge da necessidade de um objeto agregado fornecer um meio de acessar seus elementos sem expor a sua estrutura interna e também de que, muitas vezes, é necessário percorrer a coleção de maneiras diferentes ou manter mais de um percurso simultâneo, mas expandir a interface do agregado com métodos específicos para cada tipo de travessia tornaria o código confuso e rígido. O Iterator resolve esse problema ao separar a responsabilidade de acesso e iteração da coleção para um objeto específico: o iterador. A interface Iterator define os métodos para percorrer os elementos, enquanto o iterador mantém internamente a posição atual do percurso. Isso também permite que a mesma coleção seja percorrida de diferentes formas sem alterar sua implementação interna.

Onde utilizar

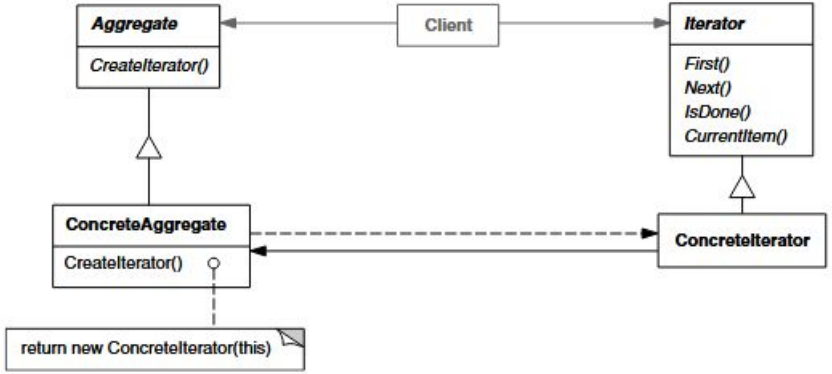
- Para acessar os conteúdos de um objeto agregado sem expor sua representação interna;
- Para suportar múltiplos percursos de objetos agregados;
- Para fornecer uma interface uniforme que percorra diferentes estruturas agregadas (ou seja, para suportar a iteração polimórfica);

Diagrama UML



Participantes

- Iterator - Define uma interface para acessar e percorrer elementos
- ConcreteIterator - Implementa a interface de Iterator
- Aggregate - Define uma interface para a criação de um objeto Iterator
- ConcreteAggregate - Implementa a interface de criação do Iterator para retornar uma instância do ConcreteIterator apropriado



Demonstração do Iterator

Domínio do Problema

O dono de uma empresa possui três lojas físicas que armazenam os produtos de formas diferentes (array, ArrayList e Hashtable) e deseja criar uma loja virtual que mostre todos os itens. Se a loja virtual acessasse diretamente os inventários, ficaria fortemente acoplada às implementações de cada loja, dificultando manutenção e adição de novas lojas. Para resolver isso, aplicamos o padrão Iterator, fazendo com que cada loja forneça seu próprio iterador através do método de criar o iterador. A loja virtual percorre os produtos uniformemente, sem conhecer a estrutura interna de cada loja, mantendo o código flexível e escalável.

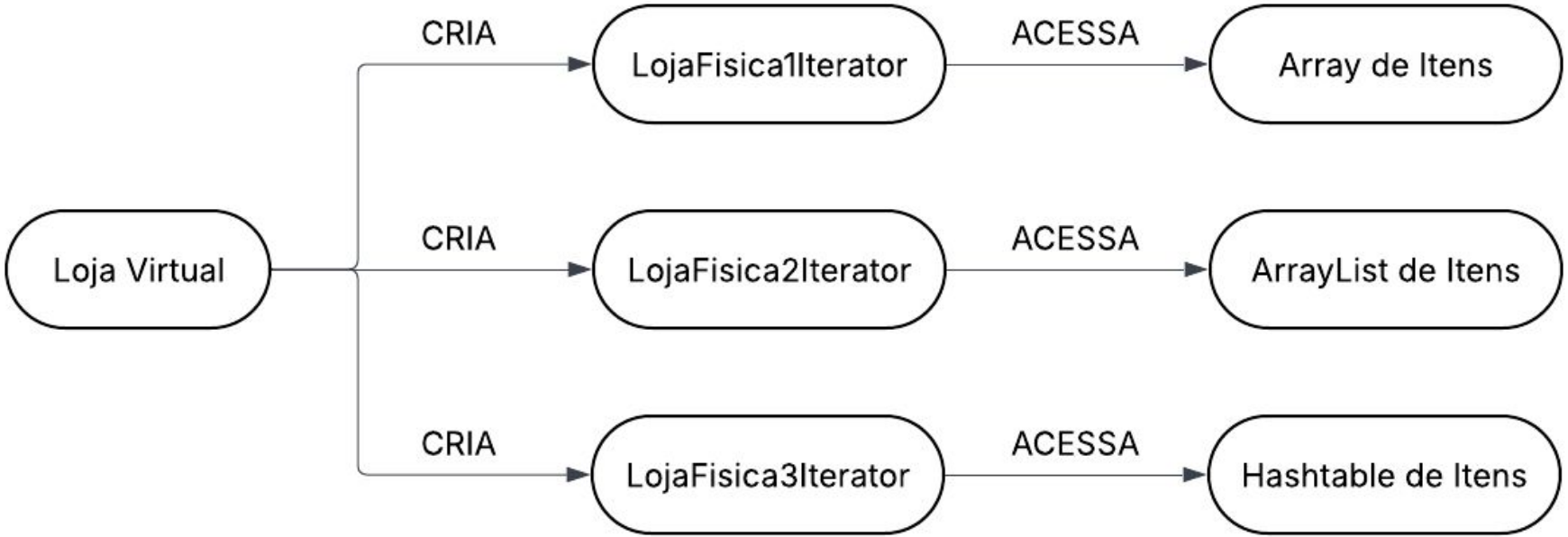
O que varia

- A forma de armazenar os produtos nas lojas físicas;

O que não varia

- A existência de um conjunto de produtos a ser percorrido e mostrado;
- A loja virtual sempre será responsável por exibir os produtos;

Diagrama da solução



Participantes

- LojaVirtual

```
public class LojaVirtual {

    static LojaFisica lojaFisica1;  2 usages
    static LojaFisica lojaFisica2;  2 usages
    static LojaFisica lojaFisica3;  2 usages

    public static void main(String[] args) {

        lojaFisica1 = new LojaFisica1();
        lojaFisica2 = new LojaFisica2();
        lojaFisica3 = new LojaFisica3();

        mostrarProdutos();

    }

    public static void mostrarProdutos(){  1 usage
        Iterator<Item> loja1iterator = lojaFisica1.createIterator();
        Iterator<Item> loja2iterator = lojaFisica2.createIterator();
        Iterator<Item> loja3iterator = lojaFisica3.createIterator();
        mostrarProdutos(loja1iterator);
        mostrarProdutos(loja2iterator);
        mostrarProdutos(loja3iterator);
    }

    public static void mostrarProdutos(Iterator<Item> iterator){  3 usages
        while(iterator.hasNext()){
            Item item = iterator.next();
            System.out.println(item.getNome()
                    + ", " + item.getPreco()
                    + ", " + item.getQuantidade());
        }
        System.out.println("");
    }

}
```

Participantes

- LojaFisica
- Iterator
- Item

```
public interface LojaFisica {  6 usages
    Iterator<Item> createIterator();
}
```

```
public interface Iterator<T> {
    boolean hasNext();  1 usage
    T next();  1 usage  3 implement
}
```

```
public class Item {  28 usages

    private String nome;  2 usages
    private double preco;  2 usages
    private int quantidade;  2 usages

    public Item(String nome, double preco, int quantidade) {
        this.nome = nome;
        this.preco = preco;
        this.quantidade = quantidade;
    }

    public String getNome() { return nome; }
    public double getPreco() { return preco; }
    public int getQuantidade() { return quantidade; }
}
```

Participantes

- LojaFisica1
- LojaFisica1Iterator

```
public class LojaFisica1Iterator implements Iterator<Item> {

    Item[] itens;  3 usages
    int position = 0;  2 usages

    public LojaFisica1Iterator(Item[] itens) {  1 usage
        this.itens = itens;
    }

    @Override  1 usage
    public boolean hasNext() {
        return position < itens.length;
    }

    @Override  1 usage
    public Item next() {
        return itens[position++];
    }

}
```

```
public class LojaFisica1 implements LojaFisica {  2 usages

    Item[] itens;  3 usages
    int numeroItens = 0;  2 usages

    public LojaFisica1() {  1 usage
        itens = new Item[5];

        adicionarItem( nome: "RTX 2050",  valor: 900,  quantidade: 54);
        adicionarItem( nome: "RTX 2060",  valor: 1100,  quantidade: 35);
        adicionarItem( nome: "Iphone 13",  valor: 5500,  quantidade: 91);
        adicionarItem( nome: "Samsung S20",  valor: 2500,  quantidade: 27);
        adicionarItem( nome: "Intel i5",  valor: 1300,  quantidade: 5);

    }

    public void adicionarItem(String nome, int valor, int quantidade) {  5 usages
        Item item = new Item(nome, valor, quantidade);
        itens[numeroItens] = item;
        numeroItens++;
    }

    public Iterator createIterator() { return new LojaFisica1Iterator(itens); }

}
```

Participantes

- LojaFisica2
- LojaFisica2Iterator

```java
import java.util.ArrayList;

public class LojaFisica2Iterator implements Iterator<Item> {

    ArrayList<Item> itens; 3 usages
    int position = 0; 2 usages

    public LojaFisica2Iterator(ArrayList<Item> itens) { 1 usa
        this.itens = itens;
    }

    @Override 1 usage
    public boolean hasNext() {
        return position < itens.size();
    }

    @Override 1 usage
    public Item next() {
        return itens.get(position++);
    }

}
```

```java
import java.util.ArrayList;

public class LojaFisica2 implements LojaFisica{ 1 usage

    ArrayList<Item> itens; 3 usages

    public LojaFisica2() { 1 usage
        itens = new ArrayList<>();

        adicionarItem( nome: "RTX 3050", valor: 1200, quantidade: 74);
        adicionarItem( nome: "RTX 3060", valor: 1700, quantidade: 25);
        adicionarItem( nome: "Iphone 14", valor: 7000, quantidade: 12);
        adicionarItem( nome: "Samsung S21", valor: 3700, quantidade: 29);
        adicionarItem( nome: "Intel i7", valor: 1700, quantidade: 11);

    }

    public void adicionarItem(String nome, int valor, int quantidade) { 5 usages
        Item item = new Item(nome, valor, quantidade);
        itens.add(item);
    }

    public Iterator createIterator() { return new LojaFisica2Iterator(itens); }

}
```

Participantes

- LojaFisica3
- LojaFisica3Iterator

```
import java.util.Hashtable;

public class LojaFisica3Iterator implements Iterator<Item> {

    Hashtable<Integer,Item> itens;
    int position = 0;

    public LojaFisica3Iterator(Hashtable<Integer,Item> itens) {
        this.itens = itens;
    }

    @Override
    public boolean hasNext() {
        return position < itens.size();
    }

    @Override
    public Item next() {
        return itens.get(position++);
    }

}
```

```
import java.util.Hashtable;

public class LojaFisica3 implements LojaFisica{

    Hashtable<Integer, Item> itens;
    int chave = 0;

    public LojaFisica3() {
        itens = new Hashtable<>();

        adicionarItem( nome: "RTX 4050", valor: 1900, quantidade: 67);
        adicionarItem( nome: "RTX 4060", valor: 2300, quantidade: 43);
        adicionarItem( nome: "Iphone 15", valor: 7000, quantidade: 3);
        adicionarItem( nome: "Samsung S22", valor: 3700, quantidade: 14);
        adicionarItem( nome: "Intel i9", valor: 1700, quantidade: 89);

    }

    public void adicionarItem(String nome, int valor, int quantidade) {
        Item item = new Item(nome, valor, quantidade);
        itens.put(chave, item);
        chave++;
    }

    public Iterator createIterator() { return new LojaFisica3Iterator(itens); }

}
```

Análise dos resultados

- Desacoplamento entre Loja Virtual e coleções internas;
- Uniformidade no acesso aos produtos;
- Extensibilidade para novas lojas;
- Responsabilidade única e modularidade;

```
/* certo */

Iterator<Item> loja1iterator = lojaFisica1.createIterator();
Iterator<Item> loja2iterator = lojaFisica2.createIterator();
Iterator<Item> loja3iterator = lojaFisica3.createIterator();

mostrarProdutos(loja1iterator);
mostrarProdutos(loja2iterator);
mostrarProdutos(loja3iterator);

/* errado */

Item[] itensLoja1 = (Item[]) lojaFisica1.getItens();
ArrayList<Item> itensLoja2 = (ArrayList<Item>) lojaFisica1.getItens();
Hashtable<Integer, Item> itensLoja3 = (Hashtable<Integer, Item>) lojaFisica1.getItens();

for(int i = 0; i < itensLoja1.length; i++) {
    System.out.println(itensLoja1[i].getNome());
}
for(Item item : itensLoja2) {
    System.out.println(item.getNome());
}
for(Integer chave : itensLoja3.keySet()) {
    System.out.println(itensLoja3.get(chave).getNome());
}
```

Demonstração