Introdução ao Framework de Coleções em Java

ENTENDENDO ARRAYLIST

Framework de Coleções

Definição de Framework de Coleções:

- Um framework de coleções em Java se refere a um conjunto de interfaces e classes que fornecem estruturas de dados flexíveis e eficientes para armazenar e manipular coleções de objetos. O objetivo principal desse framework é oferecer uma variedade de implementações de estruturas de dados comuns, como listas, conjuntos, mapas, entre outros, além de algoritmos para operar sobre essas coleções.

Importância em JAVA

Reutilização de código: O framework de coleções fornece implementações padronizadas de estruturas de dados comuns, como listas, conjuntos e mapas. Isso permite que os desenvolvedores reutilizem código existente em vez de escrever suas próprias implementações personalizadas.

Eficiência: As implementações fornecidas pelo framework de coleções são otimizadas para desempenho e eficiência. Isso significa que as operações comuns, como adicionar, remover e acessar elementos, são executadas de forma rápida e eficiente.

Flexibilidade: O framework de coleções oferece uma variedade de estruturas de dados para atender às diferentes necessidades de programação. Por exemplo, você pode escolher entre ArrayList, LinkedList ou Vector para armazenar uma lista de elementos, dependendo dos requisitos específicos do seu aplicativo.

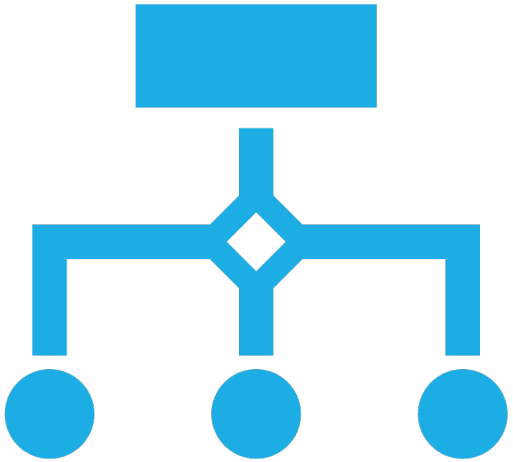
Segurança e tipo-safety: As coleções em Java são tipadas, o que significa que você pode especificar o tipo de elementos que a coleção pode conter. Isso ajuda a evitar erros de tipo em tempo de execução e torna o código mais seguro e fácil de entender.

API consistente: O framework de coleções em Java fornece uma API consistente e intuitiva para manipular coleções de elementos. Isso facilita o desenvolvimento de código claro e legível e simplifica a manutenção do código ao longo do tempo.

Suporte a iteração: O framework de coleções oferece suporte a iteração fácil e eficiente sobre os elementos de uma coleção usando loops for-each, o que simplifica o processo de processamento de dados.

Interface Collection

- Interface Collection
  - A interface Collection é a raiz da hierarquia de interfaces de coleções em Java. Ela define operações básicas comuns a todas as implementações de coleções, como adicionar elementos, remover elementos, verificar se um elemento está presente, obter o tamanho da coleção, entre outros.



Interface LIST

Definição da interface List

- A List é uma coleção ordenada que pode conter elementos duplicados. Ela permite o controle preciso sobre onde cada elemento é inserido e pode acessar elementos pelo seu índice (posição na lista).

Características da List (ordem, duplicatas)

- **Ordenada**: Os elementos na lista seguem uma sequência específica.
- **Elementos Duplicados**: A lista pode conter elementos duplicados, ou seja, elementos que são iguais.
- **Acesso por Índice**: Cada elemento tem um índice associado que pode ser usado para acessar o elemento.

Métodos específicos da List (get, set, indexOf, etc.)

- get(int index): Retorna o elemento na posição especificada na lista.
- set(int index, E element): Substitui o elemento na posição especificada na lista pelo elemento especificado.
- indexOf(Object o): Retorna o índice da primeira ocorrência do elemento especificado na lista, ou -1 se a lista não contém o elemento.
- add(int index, E element): Insere o elemento especificado na posição especificada na lista.
- remove(int index): Remove o elemento na posição especificada na lista.

Introdução ao ArrayList

O que é ArrayList?

- Um ArrayList em Java é uma implementação da interface List que utiliza um array dinâmico para armazenar elementos. Isso significa que o tamanho do ArrayList pode aumentar ou diminuir dinamicamente à medida que elementos são adicionados ou removidos, proporcionando uma maior flexibilidade em comparação com arrays tradicionais.

Quando usar ArrayList:

- Quando você precisa de uma coleção de elementos com tamanho dinâmico.
- Quando você precisa realizar operações de adição ou remoção de elementos com frequência.
- Quando você precisa trabalhar com tipos de dados genéricos (ArrayLists podem ser parametrizados com tipos específicos usando generics).
- Quando você precisa de métodos convenientes fornecidos pela classe ArrayList para manipular a coleção de elementos de forma eficiente.

Complexidade Temporal - ArrayList

Acesso: Operações de acesso, como get e set, são de tempo constante, O(1) porque elas acessam diretamente o elemento em um array interno pelo índice.

Adição: Adicionar um elemento no final da lista, usando add(E e), é uma operação de tempo constante, O(1)na maioria dos casos. No entanto, se o array interno precisar ser redimensionado (o que acontece quando a lista atinge sua capacidade máxima), a complexidade pode ser O(n)para essa operação específica, pois todos os elementos precisam ser copiados para um novo array.

Inserção/Remoção: Inserir ou remover elementos de qualquer lugar que não seja o final da lista tem uma complexidade temporal linear, O(n) porque pode exigir o deslocamento de elementos para manter a ordem da lista1.

Iteração: Iterar sobre todos os elementos de um ArrayList tem uma complexidade temporal linear, O(n), onde n é o número de elementos na lista.

Java

```java
import java.util.ArrayList;

public class ExemploArrayList {
    public static void main(String[] args) {
        // Criando um ArrayList
        ArrayList<String> lista = new ArrayList<String>();

        // Adicionando elementos
        lista.add("Elemento1");
        lista.add("Elemento2");
        lista.add("Elemento3");

        // Acessando elementos
        String elemento = lista.get(1); // Acessa o segundo elemento

        // Removendo elementos
        lista.remove("Elemento2");

        // Tamanho do ArrayList
        int tamanho = lista.size();

        // Mostrando todos os elementos do ArrayList
        for(String str : lista) {
            System.out.println(str);
        }
    }
}
```

Criando Removendo Alterando em ArrayList

Java

```java
import java.util.ArrayList;

public class ExemploArrayList {
    public static void main(String[] args) {
        // Criando um ArrayList de Strings
        ArrayList<String> lista = new ArrayList<>();

        // Adicionando elementos com o método add
        lista.add("Elemento1");
        lista.add("Elemento2");
        lista.add("Elemento3");

        // Removendo elementos com o método remove
        lista.remove("Elemento2"); // Remove pelo objeto
        lista.remove(0); // Remove pelo índice

        // Alterando elementos com o método set
        lista.set(0, "ElementoAlterado"); // Altera o elemento no índice 0

        // Obtendo o tamanho da lista com o método size
        int tamanho = lista.size(); // Retorna o tamanho atual da lista

        // Limpando todos os elementos com o método clear
        lista.clear();

        // Verificando se a lista está vazia após limpar
        boolean estaVazia = lista.isEmpty(); // Retorna true se a lista estiver vazia
    }
}
```

Criando Removendo Alterando em ArrayList

```java
// Criando um ArrayList
ArrayList<String> lista = new ArrayList<>();

// Adicionando elementos
lista.add("Maçã");
lista.add("Banana");
lista.add("Laranja");

// Adicionando elemento em uma posição específica
lista.add(1, "Morango");

// Adicionando todos os elementos de uma coleção
ArrayList<String> outrasFrutas = new ArrayList<>(Arrays.asList("Pera", "Uva"));
lista.addAll(outrasFrutas);

// Removendo elemento
lista.remove("Laranja");

// Removendo elemento em uma posição específica
lista.remove(0);

// Removendo todos os elementos de uma coleção
ArrayList<String> frutasARemover = new ArrayList<>(Arrays.asList("Banana", "Pera")
lista.removeAll(frutasARemover);
```

ArrayList

Exemplos:

add

addAll

remove

removeAll

ArrayList

set;

size;

isEmpty;

contains;

clear;

```
    // Obtendo o elemento em uma posição específica
    String elemento = lista.get(0);

    // Substituindo o elemento em uma posição específica
    lista.set(0, "Abacaxi");

    // Obtendo o número de elementos na lista
    int tamanho = lista.size();

    // Verificando se a lista está vazia
    boolean vazia = lista.isEmpty();

    // Verificando se a lista contém um elemento específico
    boolean contemLaranja = lista.contains("Laranja");

    // Limpando a lista
    lista.clear();

    // Imprimindo todos os elementos da lista
    System.out.println("Elementos da lista: " + lista);
}
```

```java
// Criando um ArrayList
ArrayList<String> lista = new ArrayList<>();

// Adicionando elementos
lista.add("Maçã");
lista.add("Banana");
lista.add("Laranja");
lista.add("Banana");

// Obtendo o índice da primeira ocorrência de "Banana"
int indicePrimeiraBanana = lista.indexOf("Banana");
System.out.println("Índice da primeira ocorrência de 'Banana': " + indicePrimeiraBanana);

// Obtendo o índice da última ocorrência de "Banana"
int indiceUltimaBanana = lista.lastIndexOf("Banana");
System.out.println("Índice da última ocorrência de 'Banana': " + indiceUltimaBanana);

// Convertendo a lista para um array
String[] array = lista.toArray(new String[0]);

// Imprimindo o array
System.out.println("Array contendo todos os elementos da lista:");
System.out.println(Arrays.toString(array));
}
```

ArrayList

indexOf;

lastIndexOf;

toArray;

Main.java | Aluno.java

```java
package up.edu.br.subclasses;

import java.util.ArrayList;
import java.util.List;

public class Main
{
    public static void main(String[] args)
    {
        List<Aluno> lista = new ArrayList<Aluno>();

        Aluno a = new Aluno( nome: "João", curso: "Engenharia", nota: 8.0);
        Aluno b = new Aluno( nome: "Maria", curso: "Medicina", nota: 9.0);
        Aluno c = new Aluno( nome: "José", curso: "Direito", nota: 7.0);

        lista.add(a);
        lista.add(b);
        lista.add(c);

        System.out.println(lista);
    }
}
```

ArrayList

Add objeto;

Main.java | Aluno.java

```java
package up.edu.br.subclasses;

public class Aluno  8 usages
{
    private String nome;  3 usages
    private String curso;  2 usages
    private double nota;  2 usages

    Aluno(String nome, String curso, double nota)  3 usages
    {
        this.nome = nome;
        this.curso = curso;
        this.nota = nota;
    }

    public String getNome()  no usages
    {
        return this.nome;
    }

    public String toString()
    {
        return "\nNome: " + this.nome + "\nCurso: " + this.curso + "\nNota: " + this.nota + "\n";
    }
```

```
[
Nome: João
Curso: Engenharia
Nota: 8.0
,
Nome: Maria
Curso: Medicina
Nota: 9.0
,
Nome: José
Curso: Direito
Nota: 7.0
]
```

ArrayList

CLASSE ALUNO;

Exercícios propostos:

1 . Crie um programa que leia 10 nomes de pessoas e os armazene em um ArrayList. Em seguida, permita que o usuário digite um nome para buscar na lista. Se o nome for encontrado, imprima a posição em que ele se encontra.

2. Crie um programa que leia 10 números inteiros e os armazene em um ArrayList. Em seguida, imprima a quantidade de números pares e ímpares no ArrayList.

3. Utilizando a estrutura ArrayList, crie um programa que armazene números inteiros pares de 0 a 10. Em seguida, faça o programa exibir a lista dos valores armazenados. Logo depois, solicite para o usuário um valor inteiro positivo menor ou igual a 10 para ser removido (Esse valor deve ser validado), após a remoção, mostre a lista atualizada.