PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
CURSO DE ENGENHARIA DE SOFTWARE

Gabriel Borges

Programação Orientada a Objetos
Professor: Marco Mangan

Relatório Acadêmico: A Interface List em Java

Este relatório apresenta uma análise da interface List do Java Collections Framework, utilizando um exemplo prático de código Java para mostrar a sua aplicação e os conceitos de Programação Orientada a Objetos que estão relacionados. O documento inclui uma descrição detalhada do exemplo, questões e um relatório das fontes consultadas, seguindo as diretrizes de trabalhos acadêmicos.

Sumário

1. Exemplo de Uso: SistemaAlunos.java

O programa que eu construí demonstra o uso da interface List em Java para gerenciar uma lista de alunos, ilustrando operações básicas como adição, remoção e verificação de elementos. Esse exemplo serve como base para a os conceitos de Programação Orientada a Objetos (POO) abordados neste relatório.

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

public class SistemaAlunos {

    public static void main(String[] args) {
        // criei uma List, mas instanciei um ArrayList (polimorfismo)
        List<String> alunos = new ArrayList<>();


        alunos.add("Gabriel");
        alunos.add("Bruno");
        alunos.add("Carla");
        alunos.add("Isabela");
        alunos.add("Thais");
        alunos.add("Fabiano");
        // mostra alunos
        System.out.println(" Lista inicial de alunos ");
        mostrarAlunos(alunos);

        // remove um aluno
        alunos.remove("Bruno");

        System.out.println("\n Após remover Bruno ");
        mostrarAlunos(alunos);

        // verifica se algum aluno está presente na lista
        String busca = "Jorge";
        if (alunos.contains(busca)) {
            System.out.println("\n" + busca + " está cadastrada na lista.");
        } else {
            System.out.println("\n" + busca + " não está na lista.");
        }

        // mostrando o total de alunos
        System.out.println("\nTotal de alunos: " + alunos.size());
    }

    // método auxiliar para exibir os alunos (usando sobrecarga)
    public static void mostrarAlunos(List<String> lista) {
        for (String nome : lista) {
            System.out.println("- " + nome);
        }
    }
}
```

2. Descrição do Exemplo: Análise de Conceitos de POO

O código SistemaAlunos.java demonstra o uso da interface List do Java Collections Framework, aplicando alguns conceitos da Programação Orientada a Objetos para gerenciar uma coleção de nomes de alunos. Este exemplo, por mais que seja simples, é funcional e ilustra um caso prático de manipulação de listas em Java.

2.1. Polimorfismo

O conceito de polimorfismo é central na declaração e instanciação da lista de alunos:

```
List<String> alunos = new ArrayList<>();
```

Nesta linha, a variável alunos é declarada como sendo do tipo interface List<String>, mas é instanciada como um objeto da classe concreta ArrayList<String>. Isso é um exemplo clássico de polimorfismo, onde uma referência de tipo mais genérico (List) aponta para um objeto de um tipo mais específico (ArrayList) que implementa essa interface. O benefício é que o código acaba se tornando flexível. Assim, caso for necessário mudar a implementação da lista (por exemplo, para LinkedList), apenas a parte da instanciação precisaria ser alterada, sem impactar o restante do código que interage com a lista através da interface List [1, 3].

2.2. Interface List

A List é uma interface do Java Collections Framework que define um contrato para coleções ordenadas de elementos, permitindo duplicatas. Ela estende a interface Collection e adiciona métodos específicos para acesso posicional, iteração e manipulação de elementos. Ao usar List como tipo de referência, o código se beneficia da abstração, focando no comportamento de uma lista sem se prender aos detalhes de sua implementação [1].

2.3. Classe Concreta ArrayList

A ArrayList é uma classe concreta que implementa a interface List. Ela é baseada em um array dinâmico, o que a torna eficiente para acesso aleatório a elementos (usando get(index)) e para adicionar elementos ao final. No entanto, inserções ou remoções no meio da lista podem ser mais custosas devido à necessidade de deslocar elementos [3].

2.4. Encapsulamento

Embora não explicitamente demonstrado com modificadores de acesso private para atributos, o uso da interface List promove um tipo de encapsulamento. O código que utiliza a lista alunos não precisa saber como os dados são armazenados internamente (ArrayList ou LinkedList); ele interage apenas com os métodos definidos pela interface List. Isso oculta os detalhes de implementação e expõe apenas a funcionalidade necessária.

2.5. Herança

A interface List estende a interface Collection, que por sua vez estende a interface Iterable. Essa hierarquia é um exemplo de herança de interfaces, onde List herda os métodos definidos em Collection e Iterable, adicionando suas próprias funcionalidades. Isso garante que todas as Lists possuam um conjunto básico de operações de coleção e sejam iteráveis [3].

2.6. Exemplo Prático de Utilização

O programa SistemaAlunos simula um gerenciamento básico de uma lista de alunos:

Criação e Adição: Uma List de String (nomes de alunos) é criada e com seis nomes.

Exibição Inicial: O método mostrarAlunos é acionado para exibir todos os alunos presentes na lista.

Remoção: Um aluno específico ("Bruno") é removido da lista, mostrando a capacidade de modificação da coleção.

Exibição Pós-Remoção: A lista é exibida novamente para mostrar o estado após a remoção.

Verificação de Existência: O método contains é utilizado para verificar a presença de um aluno ("Jorge") na lista, ilustrando uma operação comum de busca.

Contagem de Elementos: O método size() é usado para obter o número total de alunos na lista, mostrando o tamanho atual da coleção.

Este exemplo é simples, mas eficaz em demonstrar as operações básicas e os benefícios da interface List e do polimorfismo em Java, tornando o código mais robusto e fácil de manter.

3. Questões de Concurso sobre a Interface List em Java

Questão 1

Qual das seguintes classes NÃO implementa a interface List em Java?

a) ArrayList

b) LinkedList

c) HashSet

d) Vector

Resposta: c) HashSet

Justificativa: A interface List é implementada por classes como ArrayList, LinkedList e Vector. A classe HashSet, por outro lado, implementa a interface Set, que é uma coleção de elementos únicos e não ordenados, diferentemente da List que permite elementos duplicados e mantém a ordem de inserção [1,3].

Questão 2

Considerando as implementações da interface List em Java, qual delas oferece o melhor desempenho para operações de acesso aleatório (acesso a um elemento por índice)?

a) LinkedList

b) ArrayList

c) Stack

d) PriorityQueue

Resposta: b) ArrayList

ArrayList usa um array interno, então acessar um elemento pelo índice é rápido (O(1)). LinkedList é uma lista duplamente encadeada, então acessar por índice é lento (O(n)). Stack e PriorityQueue não são List e têm desempenho diferente. [1,3]

4. Relatório de Fontes Consultadas

Este relatório detalha as fontes e ferramentas utilizadas para a elaboração do trabalho sobre a interface List em Java, conforme as diretrizes acadêmicas e as normas de trabalhos. O objetivo foi garantir a conformidade do relatório e destacar os aprendizados e as dificuldades resolvidas durante o processo.

Ferramentas Utilizadas

Para a realização desta pesquisa e elaboração deste trabalho, foram utilizadas as seguintes ferramentas:

Google Search: Utilizado para buscar questões de concursos e aprofundar o entendimento sobre a Java Collections Framework e os conceitos de Programação Orientada a Objetos (POO).

Biblioteca Central da PUCRS: Fonte de consulta para o livro Java: Como Programar, de Deitel, relacionado a linguagem Java e a programação orientada a objetos.

Gemini (IA): Utilizado como apoio para esclarecer dúvidas pontuais.

Aprendizados

Durante a construção deste trabalho, aprendi muito e algumas dificuldades foram superadas:

Polimorfismo em Java: A análise do código SistemaAlunos.java e das fontes consultadas reforçou o entendimento do polimorfismo, especialmente no uso de List<String> alunos = new ArrayList<>();, mostrando a importância de programar para a interface e não para a implementação.

Diferença entre List e Set: As pesquisas e questões de concurso ajudaram a fixar que List mantém a ordem e permite duplicatas, enquanto Set armazena apenas elementos únicos, sem ordem definida.

List vs. ArrayList vs. LinkedList: O estudo destacou as diferenças de desempenho entre as três estruturas — ArrayList é mais rápido para acesso direto, enquanto LinkedList é mais eficiente em inserções e remoções, e o List mantem a ordem dos elementos e permite duplicatas com acesso por índice.

Uso de Ferramentas Digitais: O uso combinado do Google, da biblioteca da PUCRS e do Gemini auxiliou na pesquisa e na síntese das informações de forma mais eficiente.

Referências

DEITEL, Paul J.; DEITEL, Harvey M.; FURMANKIEWICZ, Edson. Java: como programar. 8. ed. São Paulo: Pearson Prentice Hall, 2010. Cap. 20.6, p. 640. Acesso em: 7 out. 2025. [1]

GOOGLE. Gemini: Disponível em: https://gemini.google.com. Acesso em: 7 out. 2025. [2]

DEITEL, Paul J.; DEITEL, Harvey M.; FURMANKIEWICZ, Edson. Java: como programar. 8. ed. São Paulo: Pearson Prentice Hall, 2010. Cap. 20.6.1, p. 641. Acesso em: 8 out. 2025. [3]

GOOGLE. Google Search: mecanismo de busca online. Disponível em: https://www.google.com. Acesso em: 8 out. 2025. [4]