Trabalho E2: Estudo sobre a Classe `ArrayList`

Aluno: *Arthur Martins Magagnin*

Disciplina: Programação Orientada a Objetos

Curso: Engenharia de Software, PUCRS

Data de Entrega: 08 de outubro de 2025

1. Classe Escolhida

A classe escolhida para este trabalho é a `java.util.ArrayList`.

2. Exemplo de Uso

O código a seguir demonstra um caso de uso prático para a classe `ArrayList`, simulando o gerenciamento de uma lista de tarefas. O programa utiliza o conceito de polimorfismo ao declarar a variável com a interface `List` e instanciar a classe concreta `ArrayList`.

Arquivo `ListaDeTarefas.java`:

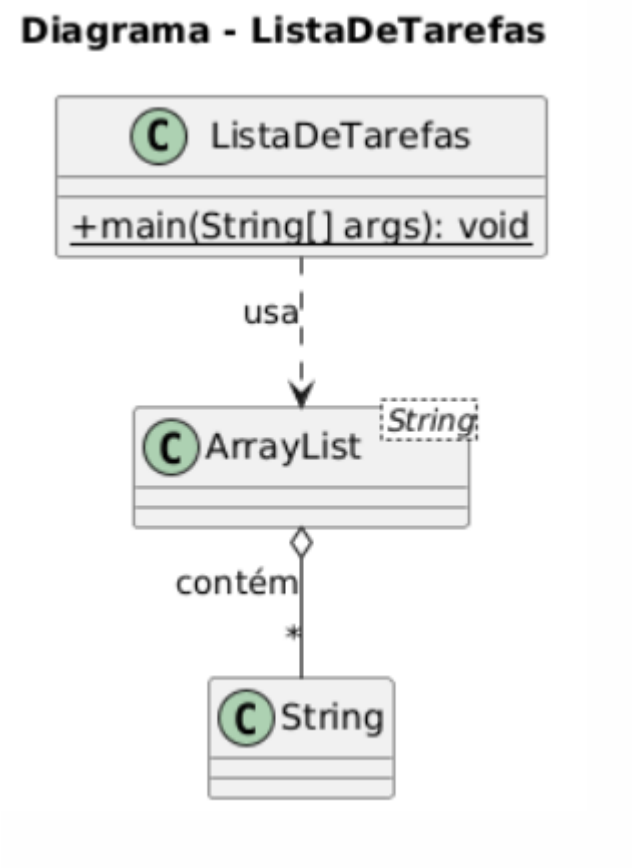
```java
/*
* Trabalho feito de forma básica e simples para mostrar o funcionamento de um ArrayList,
* utilizando o conceito de polimorfismo.
* Trabalho feito sem métodos.
**/

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

public class ListaDeTarefas {

    public static void main(String[] args) {

        List<String> listaDeTarefas = new ArrayList<>(); //cria o arraylist

        //adiciona itens ao arraylist
        listaDeTarefas.add("Estudar Java");
        listaDeTarefas.add("Fazer o trabalho de POO");
        listaDeTarefas.add("Estudar Banco de Dados");
        listaDeTarefas.add("Passear com o cachorro");

        //print o que tem dentro do arraylist
        System.out.println("\n--- Minha Lista de Tarefas ---");
        System.out.println(listaDeTarefas);
        System.out.println("Total de tarefas: " + listaDeTarefas.size()); //mostra o tamanho do arraylist

        //mostra o que tem no indice 1 do arraylist
        String segundaTarefa = listaDeTarefas.get(1);
        System.out.println("\nA segunda tarefa na lista é: '" + segundaTarefa + "'");

        //remove algo do arraylist
        System.out.println("\nRemovendo a tarefa 'Estudar Banco de Dados'...");
        listaDeTarefas.remove(2);

        //print com a lista atualizada
        System.out.println("\n--- Minha Lista de Tarefas Atualizada ---");
        System.out.println(listaDeTarefas);
        System.out.println("Total de tarefas agora: " + listaDeTarefas.size());
    }
}
```

Referências Bibliográficas (Exemplo de Citação):

- DEITEL, Paul; DEITEL, Harvey. **Java: Como Programar**. 10ª ed. Pearson Education, 2016. (Capítulo 16: Coleções).

3. Diagrama da UML

O diagrama a seguir representa as classes e relacionamentos envolvidos no exemplo de código. Ele foi simplificado para focar nas entidades principais: a classe `ListaDeTarefas`, a classe `ArrayList<String>` utilizada, e os objetos `String` contidos nela.



https://www.plantuml.com/plantuml/umla/POsxIWDX44RxUOgFhRommJf8I22D85Z4JYp6pPzryB-4cLaXXZoGp-6BkHesX6mvXpFdcvkpUfSYsOVaBIidD97D5kugRvs7XwRabNm5t9qfKNBmtJQWgByN8XUF0KlXLZanQZoAEI_3CsIOi17zDoyezWJSOjypb-Q0n9AlLlriRbzUmThQzHIVHTPq60BLGfLtVvdxepYhq8TEz6bpn8C_N1V3aybi_8CfEcCQyrBNg6ugdBIcPF_vJZGFUTsb-0i0

4. Descrição do Exemplo

O programa `ListaDeTarefas` foi desenvolvido para exemplificar o uso prático da classe `ArrayList`, demonstrando um dos princípios fundamentais da programação orientada a objetos: o **polimorfismo**. Isso é evidenciado na linha `List<String> listaDeTarefas = new ArrayList<>();`. Nesta declaração, a variável `listaDeTarefas` é do tipo da interface (`List`), enquanto o objeto instanciado é de uma classe concreta que a implementa (`ArrayList`). Essa abordagem, conhecida como "programação voltada para a interface", permite maior flexibilidade e desacoplamento da implementação específica.

A classe `ArrayList`, por sua vez, **estende** (`extends`) a **classe abstrata** (`abstract class`) `AbstractList` e **implementa** (`implements`) a interface `List`. Essa estrutura de herança e implementação garante que `ArrayList` cumpra o "contrato" definido pela interface `List`, fornecendo implementações concretas para seus métodos.

Um exemplo de **sobrescrita** (`@Override`) pode ser observado implicitamente na linha `System.out.println(listaDeTarefas)`. A saída no console é uma representação formatada da lista (ex: `[Tarefa1, Tarefa2]`), e não um endereço de memória do objeto. Isso ocorre porque a classe `ArrayList` sobrescreve o método `toString()` herdado da classe `Object` para fornecer uma visualização útil de seu conteúdo.

O conceito de **sobrecarga** (`overload`) é bem representado pelos métodos da classe `ArrayList`. O método `remove()`, por exemplo, é sobrecarregado: o código utiliza `listaDeTarefas.remove(2)`, que aceita um `int` como índice para remover um elemento em uma posição específica. Existe uma outra versão, `remove(Object o)`, que remove a primeira ocorrência de um objeto específico da lista. O compilador sabe qual método chamar com base no tipo de argumento fornecido.

Em resumo, o exemplo, embora simples, está imerso em conceitos da programação orientada a objetos, desde a instanciação de objetos de classes como `ArrayList` e `String` (que, como todas as classes em Java, são subclasses de `Object`) até a utilização de mecanismos como polimorfismo, sobrescrita e sobrecarga, que tornam o framework de coleções do Java robusto e flexível.

5. Questões sobre ArrayList

Questão 1

Qual método é usado para descobrir o número de elementos atualmente armazenados em um `ArrayList`?

a) `length()`

b) `size()`

c) `capacity()`

d) `count()`

Resposta Correta: b) Justificativa: O método `size()` é o correto para obter a quantidade de elementos em qualquer coleção que implemente a interface `Collection`, como o `ArrayList`. A propriedade `length` é usada para arrays (`[]`), `capacity()` está relacionado à capacidade interna do `ArrayList` antes de ele precisar se redimensionar, e `count()` não é um método padrão da classe.

Questão 2

Considere o código: `ArrayList<String> nomes = new ArrayList<>(); nomes.add("Ana"); nomes.add("Bia"); nomes.add("Caio");`. Qual linha de código acessa corretamente o valor "Bia"?

a) `nomes[1];`

b) `nomes.get(1);`

c) `nomes.get("Bia");`

d) `nomes.access(1);`

Resposta Correta: b) Justificativa: `ArrayList` utiliza o método `get(index)` para acessar elementos. Como a contagem de índices começa em zero, "Bia" está na posição 1. A sintaxe `nomes[1]` é usada para arrays, não para `ArrayList`. Os métodos `get("Bia")` e `access(1)` não existem na classe `ArrayList` para essa finalidade.

6. Relatório de Fontes Consultadas e Aprendizados

Fontes Consultadas

Para a realização deste trabalho, foram consultadas as seguintes fontes:

1. **Bibliografia Principal:**
    - DEITEL, Paul; DEITEL, Harvey. **Java: Como Programar**. 10ª ed. Pearson Education, 2016. (Consulta ao Capítulo 16, "Coleções", para a compreensão fundamental da estrutura e dos métodos da classe `ArrayList`).
2. **Documentação Online:**
    - **Oracle Java Documentation:** A documentação oficial da classe `java.util.ArrayList` foi utilizada como fonte primária para verificar a assinatura de métodos, as interfaces implementadas e as classes herdadas.
3. **Ferramentas de IA e Desenvolvimento:**
    - **Google Gemini:** A ferramenta foi utilizada como assistente para a geração do código de exemplo, criação do diagrama UML e auxílio na formulação das descrições textuais e questões.
    - **PlantUML:** O software online PlantUML foi utilizado para a criação e renderização do diagrama de classes UML.
    - **GitHub Codespaces:** O ambiente de desenvolvimento foi utilizado para escrever, compilar e testar o código Java.

Aprendizados e Dificuldades

O desenvolvimento deste trabalho permitiu aprofundar o conhecimento sobre a Java Collections Framework, em especial sobre a classe `ArrayList`. O principal aprendizado foi a aplicação prática do **polimorfismo** ao declarar um objeto por sua interface (`List`), consolidando a compreensão sobre como essa prática torna o código mais flexível.

Uma dificuldade inicial encontrada foi a correta sintaxe dos comandos de compilação (`javac`) e execução (`java`) via linha de comando. Superado esse obstáculo, o processo de desenvolvimento fluiu de maneira mais eficiente.