UniEinstein
CENTRO UNIVERSITÁRIO | COLÉGIO

Sobre a prova

Docente: Leandro Lopes Taveira
Coordenador: Pedro Ivo Garcia Nunes

Instruções e Regras para a Avaliação

- **Início: 19:30h - Moodle fecha as 22:05h (não tem choro)**
- **Dispositivos Eletrônicos:** O celular deve ser mantido sobre a torre do computador (CPU).
- **Conectividade:** Mantenha o cabo de rede removido do computador durante a realização de toda a prova.
- **Entrega da Prova:** A prova será enviada eletronicamente pelo Moodle.
- **Procedimento Final:** Ao concluir a prova, avise o professor antes de conectar o cabo de rede para o envio.
- **Tempo Limite para Envio:** Após a conexão, você terá um tempo máximo de 5 minutos para realizar o envio e se retirar do laboratório.
- **Intervalo:** Não haverá permissão para saídas da sala para banheiro, lanche ou água
- **Disposição:** Mantenha um computador de distância entre você e próximo colega.

Critérios de Avaliação:

- Correta implementação da estrutura escolhida, utilizando nós encadeados.
- Funcionalidade dos métodos descritos, para diferentes tipos de dados.
- Tratamento de casos de estrutura vazia.
- Implementação do menu básico e interação com o usuário.
- Organização e clareza do código.
- Uso adequado de comentários (opcional, mas recomendado).

UniEinstein
CENTRO UNIVERSITÁRIO | COLÉGIO

Dicionários e Conjuntos

Docente: Leandro Lopes Taveira
Coordenador: Pedro Ivo Garcia Nunes

Revisão - HashMap

Conceito

- Estrutura de dados que armazena **pares chave-valor**.
- Baseado em **tabela hash** para acesso O(1) em média.

Operações Básicas

- `put(chave, valor)`: Inserir/Atualizar
- `get(chave)`: Buscar
- `remove(chave)`: Remover
- `containsKey(chave)`: Verificar existência

Aplicações Comuns de HashMap

1. Contagem de Frequência

- Contar palavras em um texto
- Contar caracteres em uma string
- Contar elementos em uma lista

2. Indexação

- Índice de livros por palavra-chave
- Dicionário de sinônimos
- Catálogo de produtos por categoria

3. Cache

- Armazenar resultados de operações custosas
- Cache de páginas web
- Cache de consultas de banco de dados

4. Mapeamento de IDs

- ID do usuário → dados do usuário
- Código do produto → informações do produto

Exemplo - Contador de Palavras

```java
public class Main {
    public static void main(String[] args) {

        String frase = "Estrutura de dados é a materia mais legal";
        String[] palavras = frase.split( regex: " ");

        Map<String, Integer> contador = new HashMap<>();

        for (String palavra : palavras) {
            if (contador.containsKey(palavra)) {
                contador.put(palavra, contador.get(palavra) + 1);
            } else {
                contador.put(palavra, 1);
            }
        }
        System.out.println(contador);
    }
}
```

Métodos Úteis de HashMap

`putIfAbsent(chave, valor)`

- Insere apenas se a chave **não estiver presente**.

```
mapa.putIfAbsent("nova_chave", "valor_padrão");
```

`getOrDefault(chave, valorPadrão)`

- Retorna o valor ou um **valor padrão** se a chave não existir.

```
int count = contador.getOrDefault("palavra", 0);
```

Exercício 1: Gerenciador de Inventário Simples

Crie um programa que utilize um `HashMap<Integer, String>` para simular um inventário de produtos.

1. Adicione 4 produtos ao mapa: `(101, "Parafuso"), (102, "Martelo"), (103, "Serra"), (104, "Prego")`.
2. Busque e imprima o nome do produto com o **ID 103** usando o método `get()`.
3. Verifique se o produto com o **ID 105** existe no inventário usando `containsKey()`. Imprima o resultado.
4. Remova o produto com o **ID 104**.
5. Ao final, use um laço de repetição para **percorrer e imprimir todos os pares chave-valor** restantes no inventário.

Exercício 2: Aplicando Desconto a Produtos (Percorrer Valores)

Simule o processamento de preços de produtos para aplicar um desconto.

1. Crie um `HashMap<String, Double>` onde a chave é o nome de um produto e o valor é o preço.
2. Adicione 4 produtos e seus preços (ex: Notebook: 4500.00, Mouse: 80.00, Monitor: 1200.00).
3. Use um laço de repetição para **percorrer apenas os valores** (os preços) do mapa.
4. Dentro do laço, calcule e imprima o novo preço de cada produto após aplicar um desconto de **10%**. (Você não precisa atualizar o mapa, apenas imprimir o novo preço calculado.)

Exercício 3: Sistema de Configurações de Aplicativo

Crie um programa que simule o gerenciamento de configurações de um aplicativo. O objetivo é permitir que o usuário interaja com as configurações usando um `HashMap<String, String>`.

1. Crie um `HashMap<String, String>` chamado `configuracoes` para armazenar o nome da configuração (chave) e seu valor (valor).
2. Adicione 3 configurações iniciais: `"tema"` com valor `"dark" | "idioma"` com valor `"pt-br" | "notificacoes"` com valor `"ativadas"`
3. Crie um **menu simples** de texto no método `main` para que o usuário possa escolher entre as seguintes ações:
    - **Opção 1**: **Consultar** uma configuração. (Deve usar **`getOrDefault()`**, retornar `"NÃO DEFINIDO"` se a chave não existir).
    - **Opção 2**: **Adicionar** uma nova configuração (deve verificar se a chave já existe).
    - **Opção 3**: **Alterar** uma configuração existente (deve verificar se a chave existe antes de prosseguir).
    - **Opção 4**: Sair do programa.
4. Implemente a lógica para cada opção, utilizando os métodos apropriados do `HashMap` (`put()`, `getOrDefault()`, `containsKey()`).

Introdução a Conjuntos (Set)

Conceito

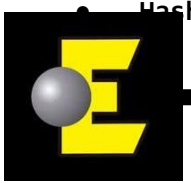
- Coleção que **não permite elementos duplicados**.
- Similar a um **conjunto matemático**.

Características

- **Elementos únicos**
- **Não há índices** (como em List)
- **Operações:** adicionar, remover, verificar existência

Implementação Comum

- **HashSet:** Baseado em tabela hash, não garante ordem.



Exemplo - HashSet

```java
public class Main {
    public static void main(String[] args) {
        Set<String> frutas = new HashSet<>();

        frutas.add("maçã");
        frutas.add("banana");
        frutas.add("maçã");    // Duplicata - não será adicionada
        frutas.add("laranja");

        System.out.println(frutas); // [banana, laranja, maçã]
        System.out.println("Tamanho: " + frutas.size()); // 3

        if (frutas.contains("banana")) {
            System.out.println("Banana está no conjunto!");
        }
        for (String fruta : frutas) {
            System.out.println("A fruta é: " + fruta);
        }
    }
}
```

Map vs Set

Característica	Map	Set
Armazena	Pares chave-valor	Apenas elementos
Chaves/Elementos	Chaves únicas	Elementos únicos
Acesso	Por chave	Por elemento
Uso	Mapeamento, dicionários	Unicidade, operações de conjunto

Implementação Interna

- `HashSet` usa internamente um `HashMap`!
- Elementos do `Set` são as **chaves** do `Map`.
- Valores do `Map` são um objeto **dummy** (constante).

Exercício 1: Identificador de Números Únicos

Crie um programa que receba uma lista de números inteiros, alguns deles duplicados. O objetivo é remover todas as duplicatas e imprimir a lista de números únicos.

1. Crie uma `List<Integer>` inicial com pelo menos 8 números, incluindo algumas duplicatas (ex: `[5, 10, 5, 20, 10, 15, 20, 25]`).
2. Crie um `HashSet<Integer>` e adicione todos os elementos da lista inicial a ele.
3. Imprima a lista original e o `HashSet` final para demonstrar que os elementos duplicados foram **automaticamente removidos**.

Exercício 2: Registro de Visitantes Únicos

Simule um sistema de registro onde apenas o primeiro acesso de cada visitante é importante.

1. Crie um `HashSet<String>` chamado `visitantesUnicos`.
2. Adicione os nomes de 5 visitantes (ex: "Maria", "João", "Pedro", "Maria", "Ana"). Inclua uma **duplicata** para teste.
3. Use um laço de repetição (`for-each`) para percorrer o conjunto (`Set`) e **imprimir apenas os nomes únicos** que foram registrados.
4. Imprima o tamanho do conjunto para confirmar que apenas os elementos únicos foram contabilizados.

Exercício 3: Verificação Rápida de Presença

Demonstre a eficiência do `HashSet` na verificação de existência (`contains()`).

1. Crie um `HashSet<String>` de nomes de funcionários (ex: "Carlos", "Fernanda", "Gustavo").
2. Defina um nome para buscar (ex: `"Gustavo"`).
3. Use o método **`contains()`** para verificar a presença do nome buscado e imprima uma mensagem indicando se ele foi encontrado ou não.
4. Tente buscar um nome que **não está** no conjunto (ex: `"Helena"`) e imprima o resultado.

Conclusão: HashMap e HashSet

Conceitos-Chave (Tabela Hash)

- Ambos são implementações baseadas em **tabela hash**.
- Oferecem desempenho de acesso, busca e inserção **O(1)** (tempo constante) em média, sendo ideais para grandes volumes de dados.

🔑 HashMap: O Mestre do Mapeamento

- **Propósito:** Armazenar e gerenciar **pares Chave-Valor**.
- **Regra Fundamental:** As **chaves** devem ser **únicas**.
- **Uso Ideal:** Dicionários, mapeamento de IDs (ID → Objeto) e caches.
- **Métodos Essenciais:** `put()`, `get()`, `remove()`.

🧩 HashSet: O Guardião da Unicidade

- **Propósito:** Garantir a **unicidade de elementos** em uma coleção.
- **Regra Fundamental:** Os **elementos** devem ser **únicos**. Duplicatas são ignoradas.
- **Uso Ideal:** Remoção de duplicatas, verificação rápida de presença (`contains()`) e operações de conjunto (intersecção, união).
- **Conexão:** Implementado internamente usando um `HashMap`, onde cada elemento é armazenado como uma chave.

revisão

Ato ou efeito de rever ou revisar; nova leitura, novo exame: revisão de provas. Local ou sala onde se revisam textos: revisão de um jornal.