ALUNA:LETÍCIA MISSENA

3.3)

```
class Aluno {
    constructor(nome, pontuacao) {
        this.nome = nome;
        this.pontuacao = pontuacao;
    }
}

class ListaDeAlunos {
    constructor() {
        this.alunos = [];
    }

    adicionarAluno(aluno) {
        this.alunos.push(aluno);
        this.alunos.sort((a, b) => b.pontuacao - a.pontuacao);
    }

    removerAluno(indice) {
        this.alunos.splice(indice, 1);
    }

    obterAlunoNaPosicao(posicao) {
        return this.alunos[posicao - 1];
    }

    obterPrimeiroAluno() {
        return this.alunos[0];
    }

    obterUltimoAluno() {
        return this.alunos[this.alunos.length - 1];
    }
}


const listaDeAlunos = new ListaDeAlunos();

listaDeAlunos.adicionarAluno(new Aluno("Rafael", 100));
listaDeAlunos.adicionarAluno(new Aluno("Paulo", 70));
listaDeAlunos.adicionarAluno(new Aluno("Ana", 60));
listaDeAlunos.adicionarAluno(new Aluno("Paulo", 50));
listaDeAlunos.adicionarAluno(new Aluno("Ana", 40));
listaDeAlunos.adicionarAluno(new Aluno("Ana", 40));

console.log("Lista de alunos:");
console.log(listaDeAlunos.alunos);


listaDeAlunos.removerAluno(1);
console.log("Lista de alunos após remover o primeiro:");
console.log(listaDeAlunos.alunos);

console.log("Aluno na posição 2:");
console.log(listaDeAlunos.obterAlunoNaPosicao(2));


console.log("Primeiro aluno:");
console.log(listaDeAlunos.obterPrimeiroAluno());
console.log("Último aluno:");
console.log(listaDeAlunos.obterUltimoAluno());
```

3.5)Exercício

```
public class Aluno {
   private String nome;
   private int matricula;


   public Aluno(String nome, int matricula) {
      this.nome = nome;
      this.matricula = matricula;
   }

   public String getNome() {
      return nome;
   }

   public void setNome(String nome) {
      this.nome = nome;
   }

   public int getMatricula() {
      return matricula;
   }

   public void setMatricula(int matricula) {
      this.matricula = matricula;
   }


   @Override
   public String toString() {
      return "Aluno [nome=" + nome + ", matricula=" + matricula + "]";
   }

   public static void main(String[] args) {
      // Exemplo de uso da classe Aluno
      Aluno aluno1 = new Aluno("João", 12345);
      System.out.println(aluno1);
   }
}
```

4.13)Exercício

1)

```
package br.com.caelum.ed.vetores;

import br.com.covlun.ed.Aluno;
public class Vetor {
   private Aluno[] alunos;
   private int tamanho;

   public Vetor(int capacidade) {
      this.alunos = new Aluno[capacidade];
      this.tamanho = 0;
   }

   public void adiciona(Aluno aluno) {
      aumentaCapacidade();
      this.alunos[this.tamanho] = aluno;
      this.tamanho++;
   }

   public void adiciona(int posicao, Aluno aluno) {
      if (posicao < 0 || posicao > this.tamanho) {
         throw new IllegalArgumentException("Posição inválida");
```

```
        }

        aumentaCapacidade();

        for (int i = this.tamanho - 1; i >= posicao; i--) {
            this.alunos[i + 1] = this.alunos[i];
        }

        this.alunos[posicao] = aluno;
        this.tamanho++;
    }

    public Aluno pega(int posicao) {
        if (posicao < 0 || posicao >= this.tamanho) {
            throw new IllegalArgumentException("Posição inválida");
        }

        return this.alunos[posicao];
    }

    public void remove(int posicao) {
        if (posicao < 0 || posicao >= this.tamanho) {
            throw new IllegalArgumentException("Posição inválida");
        }

        for (int i = posicao; i < this.tamanho - 1; i++) {
            this.alunos[i] = this.alunos[i + 1];
        }

        this.tamanho--;
    }

    public boolean contem(Aluno aluno) {
        for (int i = 0; i < this.tamanho; i++) {
            if (this.alunos[i].equals(aluno)) {
                return true;
            }
        }
        return false;
    }

    public int tamanho() {
        return this.tamanho;
    }

    private void aumentaCapacidade() {
        if (this.tamanho == this.alunos.length) {
            Aluno[] novosAlunos = new Aluno[this.alunos.length * 2];
            for (int i = 0; i < this.tamanho; i++) {
                novosAlunos[i] = this.alunos[i];
            }
            this.alunos = novosAlunos;
        }
    }
}
```

2)

```
package br.com.caelum.ed.vetores.testes;

import br.com.covlun.ed.Aluno;
import br.com.caelum.ed.vetores.Vetor;

public class TesteAdicionaNoFinal {
    public static void main(String[] args) {
        Aluno aluno1 = new Aluno();
        Aluno aluno2 = new Aluno();
```

```
        aluno1.setNome("Rafael");
        aluno2.setNome("Paulo");

        Vetor lista = new Vetor();
        lista.adiciona(aluno1);
        lista.adiciona(aluno2);

        Aluno alunoDoVetor = lista.pega(0);
        Aluno aluno2DoVetor = lista.pega(1);

        System.out.println(alunoDoVetor);
        System.out.println(aluno2DoVetor);
    }
}
package br.com.caelum.ed.vetores.testes;

import br.com.covlun.ed.Aluno;
import br.com.caelum.ed.vetores.Vetor;

public class TesteRemovePorPosicao {
    public static void main(String[] args) {
        Aluno aluno = new Aluno();
        Aluno aluno2 = new Aluno();

        aluno.setNome("Rafael");
        aluno2.setNome("Paulo");

        Vetor lista = new Vetor();
        lista.adiciona(aluno);
        lista.adiciona(aluno2);

        lista.remove(0);

        System.out.println(lista);
    }
}
package br.com.caelum.ed.vetores.testes;

import br.com.covlun.ed.Aluno;
import br.com.caelum.ed.vetores.Vetor;

public class TesteContemAluno {
    public static void main(String[] args) {
        Aluno aluno = new Aluno();
        Aluno aluno2 = new Aluno();

        aluno.setNome("Rafael");
        aluno2.setNome("Paulo");

        Vetor lista = new Vetor();
        lista.adiciona(aluno);
        lista.adiciona(aluno2);

        System.out.println(lista.contem(aluno));
        System.out.println(lista.contem(aluno2));

        Aluno aluno3 = new Aluno();
        aluno3.setNome("Ana");
        System.out.println(lista.contem(aluno3));
    }
}
package br.com.caelum.ed.vetores.testes;

import br.com.covlun.ed.Aluno;
import br.com.caelum.ed.vetores.Vetor;
```

```
public class TesteTamanhoLista {
   public static void main(String[] args) {
      Aluno aluno = new Aluno();
      Aluno aluno2 = new Aluno();

      aluno.setNome("Rafael");
      aluno2.setNome("Paulo");

      Vetor lista = new Vetor();
      lista.adiciona(aluno);
      lista.adiciona(aluno2);

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

3)

```
package br.com.caelum.ed.vetores;

import br.com.covlun.ed.Aluno;

public class Vetor {
   private Aluno[] alunos;
   private int totalDeAlunos;

   public Vetor(int capacidade) {
      this.alunos = new Aluno[capacidade];
      this.totalDeAlunos = 0;
   }

   @Override
   public String toString() {
      if (this.totalDeAlunos == 0) {
         return "[]";
      }

      StringBuilder builder = new StringBuilder();
      builder.append("[");

      for (int i = 0; i < this.totalDeAlunos - 1; i++) {
         builder.append(this.alunos[i]);
         builder.append(", ");
      }

      builder.append(this.alunos[this.totalDeAlunos - 1]);
      builder.append("]");

      return builder.toString();
   }
}
Vetor lista = new Vetor(10);

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

4)

```
package br.com.caelum.ed.vetores;

import br.com.caelum.ed.Aluno;

public class Vetor {
   private Aluno[] alunos = new Aluno[100000];
   private int totalDeAlunos = 0;

   public void adiciona(Aluno aluno) {
      for (int i = 0; i < this.alunos.length; i++) {
         if (this.alunos[i] == null) {
            this.alunos[i] = aluno;
```

```
            this.totalDeAlunos++;
            break;
         }
      }
   }
}
Package br.com.caelum.ed.vetores.testes;

import br.com.caelum.ed.Aluno;
import br.com.caelum.ed.vetores.Vetor;

public class TesteLinearVsConstante {
   public static void main(String[] args) {
      Vetor vetor = new Vetor();
      long inicio = System.currentTimeMillis();

      for (int i = 0; i < 100000; i++) {
         Aluno aluno = new Aluno();
         vetor.adiciona(aluno);
      }

      long fim = System.currentTimeMillis();
      double tempo = (fim - inicio) / 1000.0;

      System.out.println("Tempo em segundos: " + tempo);
   }
}

5)package br.com.caelum.ed.vetores;

import br.com.covlun.ed.Aluno;

public class Vetor {
   private Aluno[] alunos = new Aluno[100000];
   private int totalDeAlunos = 0;



   public int tamanho() {
      return this.totalDeAlunos;
   }
}

6)package br.com.caelum.ed.vetores;

import br.com.covlun.ed.Aluno;

public class Vetor {
   private Aluno[] alunos = new Aluno[100000];
   private int totalDeAlunos = 0;

   public boolean conten(Aluno aluno) {
      for (int i = 0; i < this.alunos.length; i++) {
         if (aluno == this.alunos[i]) {
            return true;
         }
      }
      return false;
   }

 }
package br.com.caelum.ed.vetores.testes;

import br.com.covlun.ed.Aluno;
import br.com.caelum.ed.vetores.Vetor;
```

```
public class TesteTempoDoConten {
   public static void main(String[] args) {
      Vetor vetor = new Vetor();
      long inicio = System.currentTimeMillis();

          for (int i = 0; i < 100000; i++) {
         Aluno aluno = new Aluno();
         vetor.adiciona(aluno);
      }


      for (int i = 0; i < 130000; i++) {
         Aluno aluno = new Aluno();
         if (!vetor.conten(aluno)) {
            System.out.println("Erro: aluno não foi adicionado.");
         }
      }

           for (int i = 1; i < 30000; i++) {
         Aluno aluno = new Aluno();
         if (vetor.conten(aluno)) {
            System.out.println("Erro: aluno foi adicionado.");
         }
      }

      long fim = System.currentTimeMillis();
      double tempo = (fim - inicio) / 1000.0;
      System.out.println("Tempo em segundos: " + tempo);
   }
}

7)package br.com.caelum.ed.vetores;

import br.com.covlun.ed.Aluno;

public class Vetor {
   private Aluno[] alunos = new Aluno[100000];
   private int totalDeAlunos = 0;

   public Aluno pega(int posicao) {
      return this.alunos[posicao];
   }



   private boolean posicaoOcupada(int posicao) {
      return posicao >= 0 && posicao < this.totalDeAlunos;
   }
}
package br.com.caelum.ed.vetores.testes;

import br.com.covlun.ed.Aluno;
import br.com.caelum.ed.vetores.Vetor;

public class Teste {
   public static void main(String[] args) {
      Aluno aluno = new Aluno();
      aluno.setNome("Rafael");

      Vetor lista = new Vetor();
      lista.adiciona(aluno);


      Aluno alunoNaPosicao0 = lista.pega(0);
      Aluno alunoNaPosicao1 = lista.pega(1);
      Aluno alunoNaPosicao200000000 = lista.pega(200000000);
```

```
        System.out.println(alunoNaPosicao0); // Deve imprimir o nome do aluno
        System.out.println(alunoNaPosicao1); // Deve imprimir null (posição vazia)
        System.out.println(alunoNaPosicao200000000); // Deve gerar um erro (posição inexistente)
    }
}
package br.com.caelum.ed.vetores;

import br.com.covlun.ed.Aluno;

public class Vetor {
    private Aluno[] alunos = new Aluno[100000];
    private int totalDeAlunos = 0;

    public Aluno pega(int posicao) {
        if (!this.posicaoOcupada(posicao)) {
            throw new IllegalArgumentException("Posição inválida");
        }
        return this.alunos[posicao];
    }



    private boolean posicaoOcupada(int posicao) {
        return posicao >= 0 && posicao < this.totalDeAlunos;
    }
}

8)package br.com.caelum.ed.vetores;

import br.com.covlun.ed.Aluno;

public class Vetor {
    private Aluno[] alunos = new Aluno[100000];
    private int totalDeAlunos = 0;

    public void adiciona(int posicao, Aluno aluno) {
        if (!this.posicaoValida(posicao)) {
            throw new IllegalArgumentException("Posição inválida");
        }

        for (int i = this.totalDeAlunos; i > posicao; i--) {
            this.alunos[i] = this.alunos[i - 1];
        }

        this.alunos[posicao] = aluno;
        this.totalDeAlunos++;
    }

    private boolean posicaoValida(int posicao) {
        return posicao >= 0 && posicao <= this.totalDeAlunos;
    }

 }

9)public void remove(int posicao) {
    if (!this.posicaoOcupada(posicao)) {
        throw new IllegalArgumentException("Posição inválida");
    }

    for (int i = posicao; i < this.totalDeAlunos - 1; i++) {
        this.alunos[i] = this.alunos[i + 1];
    }

    this.totalDeAlunos--;
}
```

10)
```
package br.com.caelum.ed.vetores;

import br.com.caelum.ed.Aluno;

public class TesteEstoura {
    public static void main(String[] args) {
        Vetor vetor = new Vetor();
        for (int i = 0; i < 1100001; i++) {
            Aluno aluno = new Aluno();
            vetor.adiciona(aluno);
        }
    }
}
```

```
package br.com.caelum.ed.vetores;

import br.com.caelum.ed.Aluno;

public class Vetor {
    private Aluno[] alunos = new Aluno[100000];
    private int totalDeAlunos = 0;

    public void adiciona(Aluno aluno) {
        this.garantaEspaco();

    }

    public void adiciona(int posicao, Aluno aluno) {
        this.garantaEspaco();

    }

    private void garantaEspaco() {
        if (this.totalDeAlunos == this.alunos.length) {
            Aluno[] novaArray = new Aluno[this.alunos.length * 2];
            for (int i = 0; i < this.alunos.length; i++) {
                novaArray[i] = this.alunos[i];
            }
            this.alunos = novaArray;
        }
    }

}
```

11)
```
package br.com.caelum.ed.vetores;

public class Vetor<T> {
    private T[] elementos;
    private int tamanho;

    public Vetor(int capacidade) {
        this.elementos = (T[]) new Object[capacidade];
        this.tamanho = 0;
    }

    public void adiciona(T elemento) {
        aumentaCapacidade();
        this.elementos[this.tamanho] = elemento;
        this.tamanho++;
    }

    public void adiciona(int posicao, T elemento) {
        if (posicao < 0 || posicao > this.tamanho) {
            throw new IllegalArgumentException("Posição inválida");
        }
```

```
        aumentaCapacidade();

        for (int i = this.tamanho - 1; i >= posicao; i--) {
            this.elementos[i + 1] = this.elementos[i];
        }

        this.elementos[posicao] = elemento;
        this.tamanho++;
    }

    public T pega(int posicao) {
        if (posicao < 0 || posicao >= this.tamanho) {
            throw new IllegalArgumentException("Posição inválida");
        }

        return this.elementos[posicao];
    }

    public void remove(int posicao) {
        if (posicao < 0 || posicao >= this.tamanho) {
            throw new IllegalArgumentException("Posição inválida");
        }

        for (int i = posicao; i < this.tamanho - 1; i++) {
            this.elementos[i] = this.elementos[i + 1];
        }

        this.tamanho--;
    }

    public boolean contem(T elemento) {
        for (int i = 0; i < this.tamanho; i++) {
            if (this.elementos[i].equals(elemento)) {
                return true;
            }
        }
        return false;
    }

    public int tamanho() {
        return this.tamanho;
    }

    private void aumentaCapacidade() {
        if (this.tamanho == this.elementos.length) {
            T[] novosElementos = (T[]) new Object[this.elementos.length * 2];
            for (int i = 0; i < this.tamanho; i++) {
                novosElementos[i] = this.elementos[i];
            }
            this.elementos = novosElementos;
        }
    }
}
```

12)
```
Vetor<String> vetorAlunos = new Vetor<>(1000);

for (int i = 1; i <= 1000; i++) {
    vetorAlunos.adiciona("Aluno " + i);
}

System.out.println("Tamanho do vetor de alunos: " + vetorAlunos.tamanho());

Vector<String> vectorAlunos = new Vector<>(vetorAlunos.tamanho());

for (int i = 0; i < vetorAlunos.tamanho(); i++) {
```

```
    vectorAlunos.add(vetorAlunos.pega(i));
}

System.out.println("Tamanho do novo vetor (Vector) de alunos: " + vectorAlunos.size());

ArrayList<String> arrayListAlunos = new ArrayList<>(vectorAlunos);

System.out.println("Tamanho do novo vetor (ArrayList) de alunos: " + arrayListAlunos.size());
```

4.14)Exercício

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

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


        listaDeStrings.add("Olá");
        listaDeStrings.add("Mundo");

             listaDeStrings.remove("Olá");

            String primeiroElemento = listaDeStrings.get(0);
        System.out.println("Primeiro elemento: " + primeiroElemento);
    }
}
public void removeAllOccurrences(List<String> lista, String elemento) {
    lista.removeIf(e -> e.equals(elemento));
}
public void clearList(List<String> lista) {
    lista.clear();
}
public int indexOfElement(List<String> lista, String elemento) {
    return lista.indexOf(elemento);
}
public int lastIndexOfElement(List<String> lista, String elemento) {
    return lista.lastIndexOf(elemento);
}


2)import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<String> listaDeStrings = new ArrayList<>();
        listaDeStrings.add("Olá");
        listaDeStrings.add("Mundo");
        listaDeStrings.add("Olá");

            removeAllOccurrences(listaDeStrings, "Olá");

        for (String elemento : listaDeStrings) {
            System.out.println(elemento);
        }
    }

    public static void removeAllOccurrences(List<String> lista, String elemento) {
            lista.removeIf(e -> e.equals(elemento));
    }
}
```

3)
```
import java.util.ArrayList;
import java.util.List;

public class Main {
   public static void main(String[] args) {
      List<String> listaDeStrings = new ArrayList<>();
      listaDeStrings.add("Olá");
      listaDeStrings.add("Mundo");


      clearList(listaDeStrings);

            if (listaDeStrings.isEmpty()) {
         System.out.println("A lista está vazia.");
      } else {
         System.out.println("A lista ainda contém elementos.");
      }
   }

   public static void clearList(List<String> lista) {
      lista.clear();
   }
}
```

4)
```
import java.util.ArrayList;
import java.util.List;

public class Main {
   public static void main(String[] args) {
      List<String> listaDeStrings = new ArrayList<>();
      listaDeStrings.add("Olá");
      listaDeStrings.add("Mundo");
      int indice = indexOfElement(listaDeStrings, "Mundo");

      if (indice != -1) {
         System.out.println("Índice da primeira ocorrência de 'Mundo': " + indice);
      } else {
         System.out.println("'Mundo' não encontrado na lista.");
      }
   }

   public static int indexOfElement(List<String> lista, String elemento) {
      return lista.indexOf(elemento);
   }
}
```

5)
```
import java.util.ArrayList;
import java.util.List;

public class Main {
   public static void main(String[] args) {
      List<String> listaDeStrings = new ArrayList<>();
      listaDeStrings.add("Olá");
      listaDeStrings.add("Mundo");
      listaDeStrings.add("Olá");
      int indice = lastIndexOfElement(listaDeStrings, "Olá");

      if (indice != -1) {
         System.out.println("Índice da última ocorrência de 'Olá': " + indice);
      } else {
         System.out.println("'Olá' não encontrado na lista.");
      }
   }
```

```
    public static int lastIndexOfElement(List<String> lista, String elemento) {
        return lista.lastIndexOf(elemento);
    }
}
```

6) A análise amortizada é uma técnica usada para analisar o desempenho de algoritmos e estruturas de dados em múltiplas operações, em vez de focar apenas em uma única operação. É particularmente útil quando se trata de operações cujo tempo de execução pode variar significativamente, como a inserção em estruturas de dados dinâmicas, como listas ou vetores dinâmicos (arrays dinâmicos).

No caso em que um vetor dinâmico dobra sua capacidade quando fica cheio, isso é chamado de "redimensionamento". A opção de capacidade dupla é geralmente melhor que a opção incremental pelos seguintes motivos:

***Reduzir a frequência de redimensionamento*: Quando a capacidade é duplicada, a estrutura de dados será redimensionada com menos frequência do que quando apenas aumentamos a capacidade. Isso ocorre porque quando dobramos, nossa capacidade aumenta exponencialmente, o que significa que podemos realizar múltiplas inserções antes que a próxima operação de redimensionamento seja necessária. Em contrapartida, com uma estratégia incremental, a estrutura precisa ser redimensionada com mais frequência.**

***Redimensionar é menos dispendioso*: Redimensionar uma estrutura de dados pode ser caro, pois pode envolver a alocação de uma nova área de memória, a cópia de elementos existentes para a nova área de memória e, em seguida, a liberação da área de memória anterior. Quando a capacidade é duplicada, embora ainda tenhamos esse custo no redimensionamento, ele é distribuído por múltiplas operações de inserção.**
Isso significa que o custo médio por inserção é menor que o custo se redimensionamos com mais frequência.

***Complexidade assintótica*: duplicar a capacidade resulta em uma complexidade assintótica amortizada de O(1) para inserções porque o custo de redimensionamento é amortizado em múltiplas inserções. Em contraste, a estratégia incremental pode ter baixa complexidade assintótica de inserção porque o redimensionamento mais frequente pode resultar em uma complexidade de inserção de O(n), onde n é o número total de elementos.**

Portanto, ao escolher entre duplicar a capacidade e incrementar, a análise de amortização mostra que a duplicação da capacidade geralmente leva a um melhor desempenho em termos de tempo médio de execução, especialmente quando se insere um grande número de elementos.