**FACULDADE FACY WYDEN**

**MATHEUS MOURA COELHO**

**ATIVIDADE AVALIATIVA – DESENVOLVIMENTO DE SOFTWARE**

**BELÉM**

**2022**

**MATHEUS MOURA COELHO**

**ATIVIDADE AVALIATIVA – DESENVOLVIMENTO DE SOFTWARE**

Trabalho acadêmico apresentado à disciplina de Desenvolvimento de Software do Curso de Análise e Desenvolvimento de Sistemas da faculdade Faci Wyden. Requerido pelo prof. Heleno Cardoso da Silva Filho.

**BELÉM**

**2022**

**RESUMO**

Este trabalho tem como objetivo apresentar e discorrer sobre os tópicos referentes a avaliação da disciplina de Desenvolvimento de Software. Foi feita uma pesquisa abrangente acerca de todos os tópicos, e todas as implementações requeridas foram desenvolvidas

**SUMÁRIO**

[1. INTRODUÇÃO 4](#_Toc106376249)

[1.1. TÓPICOS 4](#_Toc106376250)

[1.1.1. Arquitetura Web: Fundamentos; Modelos Arquiteturais 4](#_Toc106376251)

[1.1.2. HTML: Exemplo de construção de cadastro de formulário 7](#_Toc106376252)

[1.1.3. Descrever as diferenças entre Servlets e Scripts CGI 8](#_Toc106376253)

[1.1.4. Introdução ao JSP (Java Server Pages) 9](#_Toc106376254)

[1.1.5. Introdução aos Padrões de Projetos; explicar funcionamento arquitetura MVC 10](#_Toc106376255)

[1.1.6. Introdução ao Orientação a Objetos: Classes; objeto; atributos/propriedades; métodos; níveis de visibilidade; herança; encapsulamento; polimorfismo; extends; implements 12](#_Toc106376256)

[1.1.7. Introdução a linguagem UML 15](#_Toc106376257)

[1.1.8. Modelar um domínio (livre escolha), apresentar caso de uso, diagrama de classe e sequência 16](#_Toc106376258)

[1.1.9. Introdução a JSF (Java Server Faces) 19](#_Toc106376259)

[1.1.10. Apresentação dos conceitos ORM (Object Relacional Mapping) e JPA e Hibernate 20](#_Toc106376260)

[1.1.11. Introdução a Estrutura de Dados: Tipos de Listas; Fila; Pilha; apresentar uma estrutura de dados implementada em JAVA (TED) 21](#_Toc106376261)

[1.1.12. Apresentar conceitos de estrutura de dados árvore AVL e um exemplo de implementação em JAVA 23](#_Toc106376262)

[1.1.13. Introdução de estrutura de dados grafos e exemplos 24](#_Toc106376263)

[1.1.14. Descrição da problemática do caminho com custo mínimo; problema do caixeiro viajante, algoritmo Dijkstra 25](#_Toc106376264)

[1.1.15. Caracterização da linguagem JAVA: técnicas de programação (tipos primitivos; variáveis; constantes; operadores; estruturas de seleção; estruturas de repetição; arrays; functions) 26](#_Toc106376265)

[1.1.16. Apresentar um exemplo de backend em JAVA utilizando STS Spring Boot 32](#_Toc106376266)

[1.1.17. Resolver 03 exercícios de uma das listas em JAVA: PUCRS ou UFBA 33](#_Toc106376267)

[2. CONSIDERAÇÕES FINAIS 34](#_Toc106376268)

[3. REFERÊNCIAS 35](#_Toc106376269)

1. INTRODUÇÃO

Este estudo refere-se a à disciplina de Desenvolvimento de Software, serão abordados os tópicos requeridos pelo professor Heleno Cardoso da Silva Filho.

## TÓPICOS

Este trabalho é composto por 17 tópicos, incluindo a elaboração de um vídeo apresentando um dos itens de livre escolha.

Para o vídeo em questão, foi escolhido o tópico 2 (apresentar exemplo de um formulário HTML).

### Arquitetura Web: Fundamentos; Modelos Arquiteturais

**Fundamentos**

**Navegador Web:**

O navegador age como um dispositivo genérico da interface de usuário. O usuário do aplicativo utiliza o navegador para solicitar páginas da Web: HTML ou servidor. A página retornada contém uma interface de usuário totalmente formatada, com controles de entrada e texto, que é convertida pelo navegador no modo de exibição do cliente. Todas as interações do usuário com o sistema são feitas por meio do navegador.

**Servidor Web:**

O principal ponto de acesso para todos os navegadores, são basicamente softwares responsáveis por aceitar pedidos em HTTP de clientes, geralmente os navegadores, e servi-los com respostas em HTTP, incluindo opcionalmente dados, que geralmente são páginas web, tais como documentos em HTML.

**HTTP:**

O HTTP é um protocolo que especifica como será a comunicação entre um navegador e um servidor web.

Este protocolo funciona através de um modelo computacional conhecido como cliente-servidor, onde um navegador (cliente) estabelece a comunicação com um servidor e ambos passam a trocar informações entre si.

**Página HTML:**

Uma página da Web com interface com usuário e informações de conteúdo que não passam por nenhum processamento no lado do servidor. Quando o servidor da Web recebe uma solicitação de uma página HTML, ele simplesmente recupera o arquivo e o envia para o cliente solicitante, sem filtrá-lo.

**Página de servidor:**

Páginas da Web que passam por alguma forma de processamento no lado do servidor. Em geral, essas páginas são implementadas no servidor como páginas com scripts que são processadas por meio de um filtro no servidor do aplicativo ou de módulos executáveis.

**Servidor de aplicativos:**

O servidor de aplicativo é responsável pela execução do código nas páginas de servidor. Ele pode estar localizado no mesmo equipamento do servidor da Web e também pode ser executado no mesmo espaço de processamento.

**Modelos Arquiteturais**

**Arquiteturas Monolíticas**

Esta é a arquitetura mais comum e mais utilizada para desenvolvimento de aplicações web devido a sua simplicidade e também por ser a mais antiga utilizada. De forma geral, algumas de suas características são:

Suporta diferentes tipos de cliente como desktop/mobile;

Exporta APIs para comunicação com terceiros;

Pode tratar requisições HTTP, executar regras de negócio, acessar banco de dados e trocar informações com outros sistemas.

**Arquitetura de Micro Serviços**

Este tipo de arquitetura pode oferecer quase todas as funcionalidades de uma arquitetura monolítica. Além disso, oferece maior flexibilidade e muitas outras funcionalidades, portanto, é geralmente considerada uma escolha superior para aplicações complexas. Diferente da arquitetura monolítica, é difícil generalizar esta opção pois ela pode variar muito de acordo com o caso de uso e aplicações. Em todo o caso, também existem algumas vantagens e desvantagens que devem ser analisadas na hora da decisão.

De forma geral, esta arquitetura compartilha de algumas vantagens:

Os componentes tem baixo acoplamento. Podem ser desenvolvidos, testados e escalados de forma independente;

São mais leves e com funcionalidades especificas. Um serviço de autenticação, por exemplo, só sabe fazer autenticação;

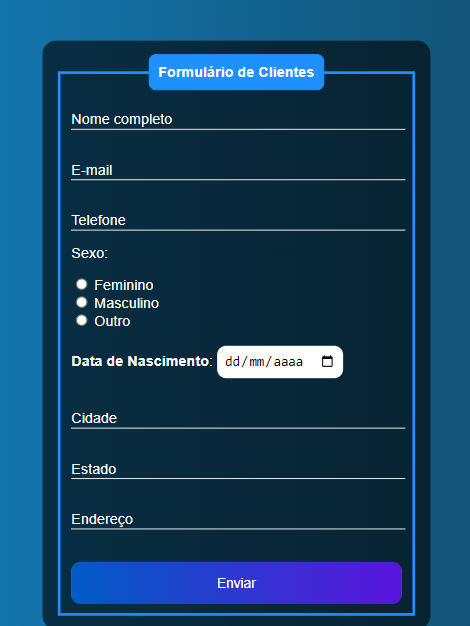
Os componentes podem ser desenvolvidos cada um com um stack de tecnologias própria;

Geralmente tem um longo e extenso setup de monitoramento e troubleshooting.

### HTML: Exemplo de construção de cadastro de formulário

O formulário de cadastro foi desenvolvido utilizando HTML e CSS, o exemplo do formulário e o código podem ser acessados através do respectivo link: <https://github.com/mthscoelho/wyden-desenv.software/tree/main/Tópico%202%20-%20Formulário>

Figura 1 – Formulário desenvolvido utilizando HTML, CSS.



Vídeo explicativo: <https://youtu.be/1wATUdzhf10>

### Descrever as diferenças entre Servlets e Scripts CGI

Servlets e Scripts CGI são programas que se encontram no servidor da web ou do aplicativo e ajudam na comunicação entre o servidor da web e o navegador (lado do cliente) para gerar o conteúdo da web dinamicamente. Servlets e Scripts CGI podem ser diferenciados porque funcionam de maneiras diferentes e possuem funcionalidades e recursos distintos. Os programas CGI (Common Gateway Interface) podem ser projetados no sistema operacional nativo e mantidos em um diretório específico. Por outro lado, o Servlet é um componente da web que geralmente é escrito em Java e executado em Java Virtual Machine (JVM).

Suas principais diferenças consistem em:

1. Scripts CGI são programas executáveis escritos no Sistema Operacional nativo do servidor. Por outro lado, os Servlets são compilados para o bytecode Java que é executado na JVM.
2. A velocidade, desempenho e eficiência do Servlet é melhor que o CGI.
3. Scripts CGI podem ser processados diretamente. De outro modo, o Servlet primeiro traduz e compila o programa e o processa.
4. O Servlet é mais seguro que o CGI, pois usa o Java.
5. O CGI é específico da plataforma, o que dificulta a alternância entre os sistemas operacionais. Por outro lado, os Servlets podem ser executados em qualquer sistema operacional que tenha instalado o JVM, portanto, ele é independente de plataforma.
6. Quando se trata de portabilidade, o Servlet é portátil, enquanto o CGI não é.

### Introdução ao JSP (Java Server Pages)

Java Server Pages (JSP) é uma linguagem de script gratuita, com especificação aberta que tem como objetivo primário a geração de conteúdo dinâmico, como arquivos HTML, DHTML, XHTML e XML, para páginas da Internet.

JSP oferece diversos benefícios como um sistema para geração de conteúdo dinâmico. Primeiramente, sendo uma tecnologia baseada em Java, ela se aproveita de todas as características e vantagens desta linguagem como orientação a objetos, herança, encapsulamento, tratamento de exceções e gerenciamento de memória automática, conduzindo assim a um código mais robusto e flexível. Além disso, tem a vantagem da portabilidade de plataforma, que permite a sua execução em diversos sistemas operacionais, como o Windows da Microsoft, Unix e Linux.

Assim, pode-se definir a JSP como uma tecnologia que provê uma maneira simples e prática de desenvolver aplicações dinâmicas baseadas em web, sendo independente de Plataforma de Sistema Operacional.

### Introdução aos Padrões de Projetos; explicar funcionamento arquitetura MVC

**Padrões de Projeto**

Padrões de projeto são soluções típicas para problemas comuns em projeto de software. Podendo ser aplicadas independentemente da linguagem. Os padrões são como plantas pré-projetadas de uma construção, que você pode alterar para resolver um problema de projeto particular em seu código.

Os padrões de projetos tornam mais fácil reutilizar soluções e arquiteturas bem sucedidas para construir softwares orientados a objetos de forma flexível e fácil de manter. O uso de padrões de projeto pode reduzir a complexidade do processo de projetar software. Além disso, o software orientado a objetos bem projetado possibilita aos projetistas reutilizar e empregar componentes preexistentes em sistemas futuros.

**Arquitetura MVC**

O MVC é um padrão de arquitetura de software responsável por contribuir na otimização da velocidade entre as requisições feitas pelo comando dos usuários.

A utilização do padrão MVC traz como benefício o isolamento das regras de negócios da lógica de apresentação, que é a interface com o usuário. Isto possibilita a existência de várias interfaces com o usuário que podem ser modificadas sem a necessidade de alterar as regras de negócios, proporcionando muito mais flexibilidade e oportunidades de reuso das classes.

Uma das características de um padrão de projeto é poder aplicá-lo em sistemas distintos. O padrão MVC pode ser utilizado em vários tipos de projetos como, por exemplo, desktop, web e mobile.

A arquitetura MVC é dividida em três componentes essenciais: Model, Controller e View.

**Model ou Modelo**

Tem a responsabilidade de gerenciar e controlar a forma como os dados se comportam por meio das funções, lógica e regras de negócios estabelecidas.

Ele é o detentor dos dados que recebe as informações do Controller, valida se ela está correta ou não e envia a resposta mais apropriada.

**Controller ou Controlador**

Responsável por intermediar as requisições enviadas pelo View com as respostas fornecidas pelo Model, processando os dados que o usuário informou e repassando para outras camadas.

**View ou Visão**

Essa camada é responsável por apresentar as informações de forma visual ao usuário. Em seu desenvolvimento devem ser aplicados apenas recursos ligados a aparência como mensagens, botões ou telas. É a parte da interface que se comunica, disponibilizando e capturando todas as informações do usuário.

**Conclusão**

Em suma, O MVC funciona como um padrão de arquitetura de software que melhora a conexão entre as camadas de dados, lógica de negócio e interação com usuário. Através da sua divisão em três componentes, o processo de programação se torna algo mais simples e dinâmico.

### Introdução ao Orientação a Objetos: Classes; objeto; atributos/propriedades; métodos; níveis de visibilidade; herança; encapsulamento; polimorfismo; extends; implements

**Introdução ao Orientação a Objetos**

Orientação a objetos é um paradigma aplicado na programação que consiste na interação entre diversas unidades chamadas de objetos. Usamos a orientação a objetos para nos basear na vida real e resolver problemas de software.

**Classe e Objeto**

Uma classe é um conjunto de características e comportamentos que definem o conjunto de objetos pertencentes à essa classe. Ela é formada por dados e comportamentos.

Para definir os dados são utilizados os atributos, e para definir o comportamento são utilizados métodos. Depois que uma classe é definida podem ser criados diferentes objetos que utilizam a classe.

A classe em si é um conceito abstrato, como um molde, que se torna concreto e palpável através da criação de um objeto. Essa criação é chamada de instanciação da classe, como se estivéssemos usando esse molde (classe) para criar um objeto.

Um objeto possui características próprias (atributos) e executa determinadas ações (métodos), sendo esses atributos e métodos provenientes da classe que origina o objeto.

**Atributos/Propriedades de Classe**

São características particulares de uma ocorrência de uma classe, como por exemplo o nome e a idade de uma pessoa.

Os dois tipos principais de propriedades são:

Estáticas: Mantém o mesmo valor durante toda a sua existência. Por exemplo, a data de nascimento de uma pessoa é uma propriedade estática, pois mantém o mesmo valor durante toda a existência do objeto instanciado.

Dinâmicas: Podem ter valores que variam com o passar do tempo. A idade de uma pessoa pode ser considerada um atributo dinâmico, pois pode variar ao longo do tempo.

**Método**

Um método é uma sequência de declarações (comandos) que possui um nome de identificação. É a lógica contida em uma classe para atribuir-lhe comportamentos. São as funções e procedimentos contidos na classe.

Toda classe possui um método especial chamado de método construtor, que inicializa o objeto instanciado. As classes também usam métodos destruidores para finalizar os objetos após seu uso.

Um método possui um nome e um corpo onde ficam os comandos que serão executados quando o método for chamado, e também podem receber dados para processamento e retornar informações.

**Níveis de Visibilidade**

Os atributos e operações de uma classe podem ser especificados para mostrar como a mesma pode ser vista e utilizada pelos outros elementos do sistema.

Existem três níveis de visibilidade:

Público: Quando definido um método ou atributo como público, é liberado o acesso a ele de qualquer lugar que contenha a classe. Representa-se colocando antes do nome do atributo ou operação o símbolo +.

Protegido: Restringe o acesso de fora da classe, porém ele ainda é acessível de suas subclasses, ou seja, as classes que herdaram os atributos dela. Representa-se colocando o símbolo #.

Privado: Mais restritivo, só da permissão de acesso a sua própria classe, em nenhum outro lugar é possível acessá-la. Somente a própria classe pode utilizar. Representa-se colocando o símbolo -.

**Herança**

Herança, em orientação a objetos, é a capacidade de um novo objeto tomar atributos e operações de um objeto existente.

É algo bastante útil para um reaproveitamento de código, pois não é necessário refazer algo que já existe. Parte-se de um ponto e se desenvolvem novos métodos.

Dessa forma, é possível criar classes mais complexas sem que seja preciso repetir código.

**Encapsulamento**

É uma das principais técnicas que define a programação orientada a objetos. Se trata de um dos elementos que adicionam segurança à aplicação em uma programação orientada a objetos pelo fato de esconder as propriedades.

Isso garante uma camada extra de proteção para a aplicação, pois os detalhes de implementação não são revelados. Ele continua funcional, mas sem mostrar como.

Sendo assim, o encapsulamento protege os dados, permitindo o acesso a eles apenas a partir de métodos específicos e autorizados.

**Polimorfismo**

O Polimorfismo possibilita a implementação de métodos em uma hierarquia de classes com a mesma assinatura e, assim, implementar um mesmo código que funcione para qualquer classe dessa hierarquia sem a necessidade de implementações específicas para cada classe. O principal objetivo do polimorfismo é diminuir a quantidade de código escrito, aumentando a clareza e a facilidade de manutenção.

**Extends**

A herança e o polimorfismo permitem a inclusão de funcionalidades em um sistema de forma muito direta. Usa-se extends quando se deseja aplicar herança à sua classe.

Por exemplo, quando falamos que a classe A estende a classe B, significa que A herda alguns (ou todos) métodos e atributos da classe B.

**Implements**

É utilizado implements quando deseja-se implementar uma interface.

Uma interface pode conter métodos e constantes. Constantes em Java são definidas pelas palavras static e final.

A interface "firma um contrato" entre classes em que define comportamentos (métodos) que devem ser sobrescritos pela classe que os herda (se essa for uma classe concreta).

### Introdução a linguagem UML

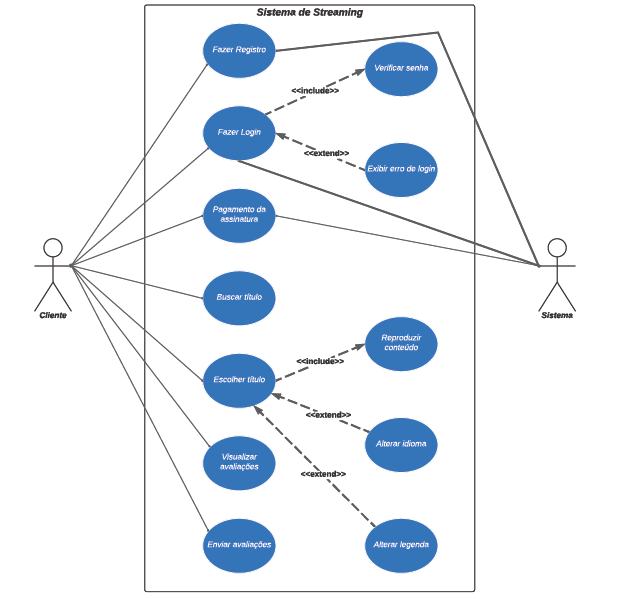
A UML permite que você “desenhe” uma “planta” do seu sistema. Essa “planta” garante, em todas as fases do projeto, seja na definição, desenvolvimento, homologação, distribuição, utilização e manutenção do mesmo, uma maior compreensão e objetividade para execução de cada ação, e, sem dúvida, quanto maior a solução, maior a necessidade de um projeto definido de forma adequada.

Desse modo, a UML é uma linguagem padrão para visualização, especificação, construção e documentação de um aplicativo ou projeto de software, e tem como objetivo aumentar a produtividade, otimizar as etapas que envolvem o desenvolvimento de um sistema, aumentando assim a qualidade do produto a ser implementado. Ela independe da ferramenta em que o aplicativo será desenvolvido. A ideia é prover uma visão lógica de todo o processo de forma a facilitar a implementação física do mesmo.

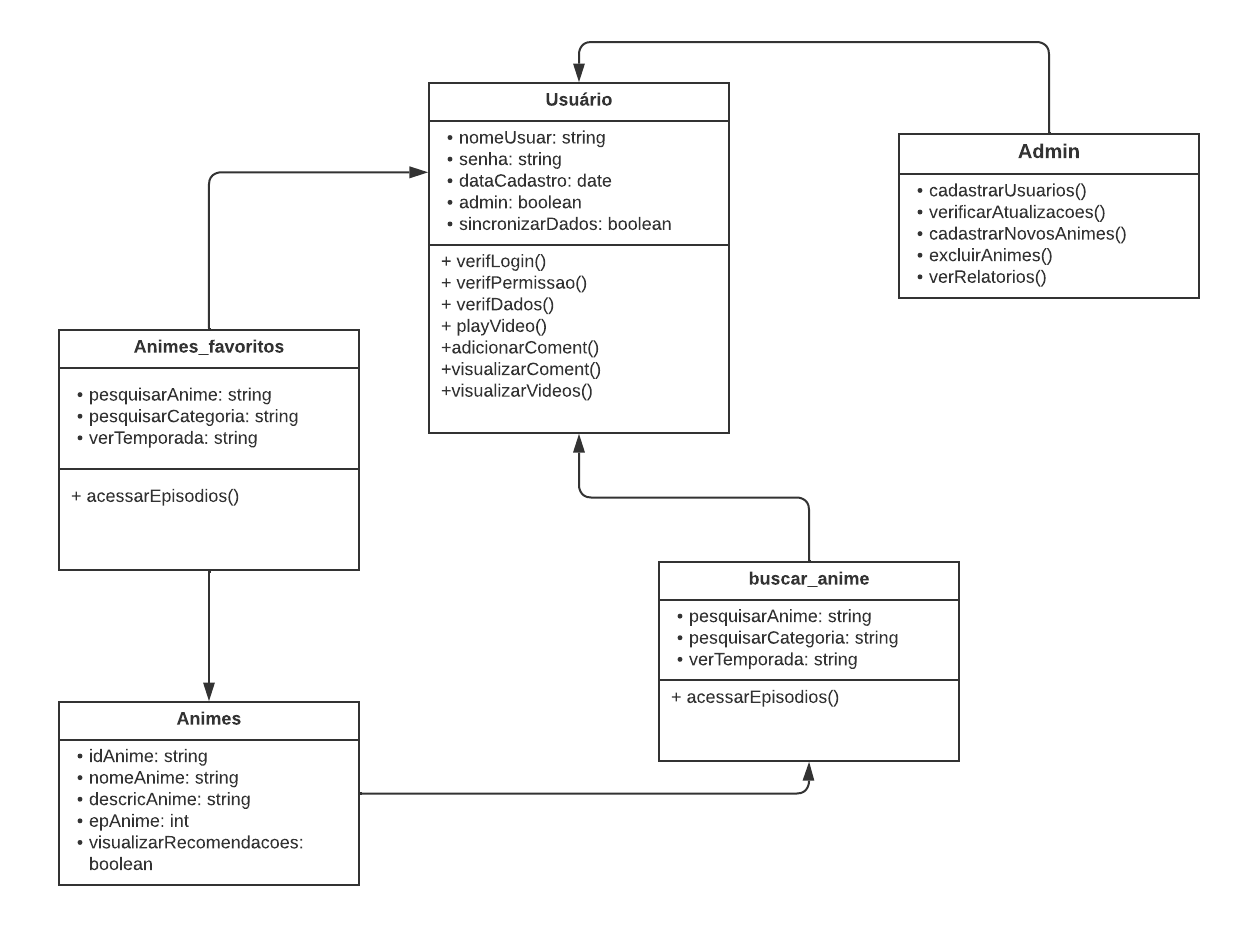
### Modelar um domínio (livre escolha), apresentar caso de uso, diagrama de classe e sequência

O domínio escolhido foi o de um Streaming de Anime e abaixo são apresentados os diagramas de caso de uso, classes e sequência para o domínio em questão.

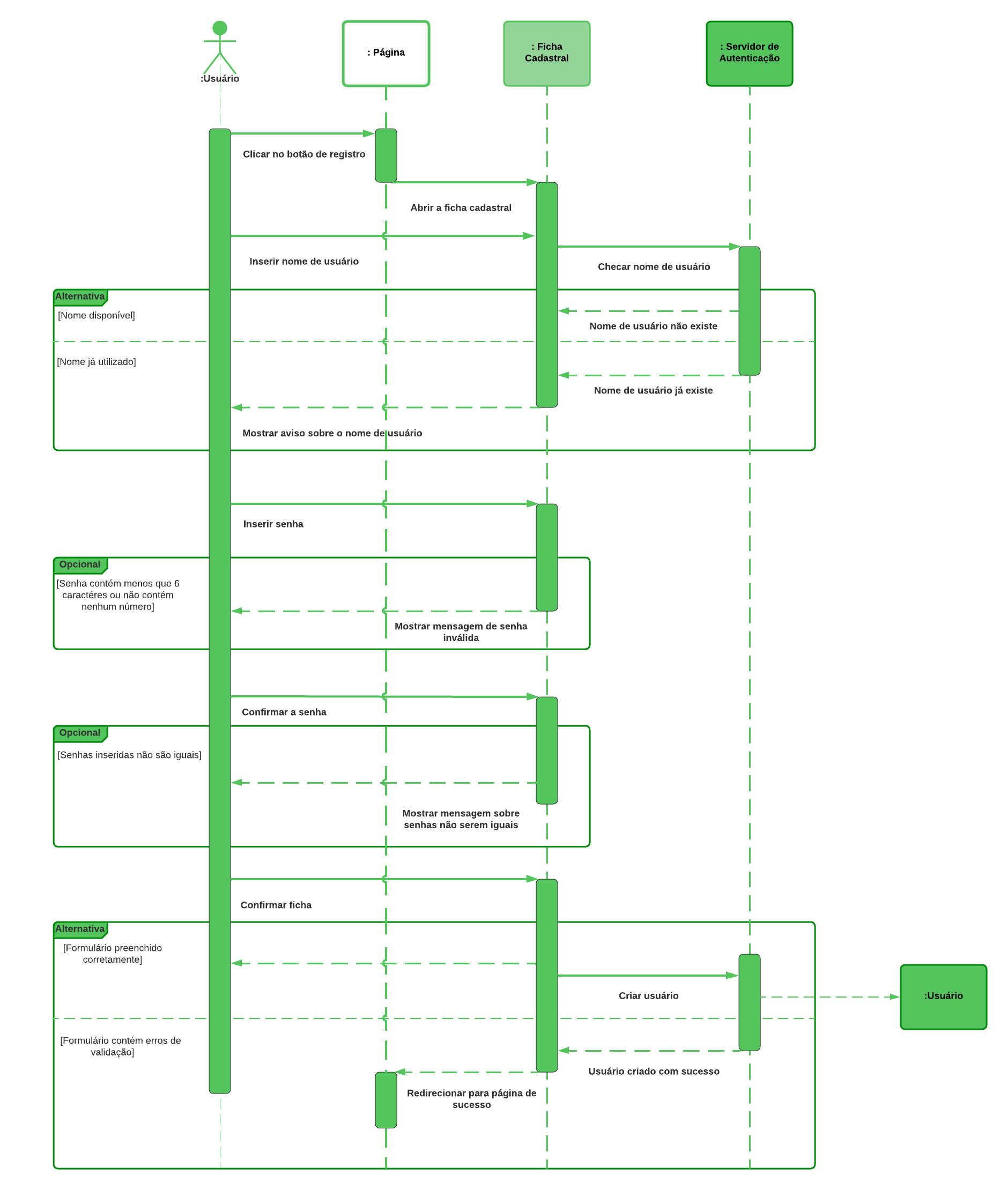
i. Diagrama de casos de uso



ii. Diagrama de classe



iii. Diagrama de sequência



### Introdução a JSF (Java Server Faces)

O Java Server Faces (JSF) é uma das tecnologias mais utilizadas para projetar, desenvolver e criar sistemas para web com a linguagem Java.

É um framework onde é efetuada a elaboração de interfaces de usuários para sistema web, colocando componentes em um formulário e ligando os a objetos Java, sendo assim, ele faz a separação entre a lógica e regras de negócio a navegação e conexões com serviços externos seguindo o modelo MVC. Tem como ponto forte a possibilidade de um grande número de componentes e um design bastante flexível.

### Apresentação dos conceitos ORM (Object Relacional Mapping) e JPA e Hibernate

**ORM**

ORM (Object Relational Mapper) é uma técnica de mapeamento objeto relacional que possibilita fazer uma relação dos objetos com os dados que os mesmos representam.

O uso da técnica de mapeamento objeto-relacional é feito através de um mapeador objeto-relacional que geralmente é a biblioteca ou framework que ajuda no mapeamento e uso do banco de dados.

As bibliotecas ou frameworks ORM definem o modo como os dados serão mapeados entre os ambientes, como serão acessados e gravados. Isso minimiza o tempo de desenvolvimento, uma vez que não é necessário desenvolver toda essa parte.

**JPA**

JPA é uma coleção de classes e métodos voltados para armazenar persistentemente as grandes quantidades de dados em um banco de dados. Vários Frameworks com base no JPA são desenvolvidos com o objetivo de proporcionar uma interação com um banco de dados relacional, evitando com que o desenvolvedor gaste tempo com o desenvolvimento de códigos voltados para a manipulação dos dados presentes no banco de dados.

Em outras palavras, o JPA fornece meios de armazenar os dados presentes nos objetos implementados no sistema desenvolvido dentro das entidades no banco de dados.

**Hibernate**

O Hibernate é uma ferramenta de mapeamento objeto-relacional (ou ORM) para Java. Basicamente como todo o ORM, o Hibernate transforma os dados da estrutura lógica de um banco de dados em objetos relacionais. Com a utilização do Hibernate, não há necessidade de escrever SQL “puro”, pois ele utiliza seu próprio código, chamado de HQL (Hibernate Query Language).

Sua principal função é abstrair o mapeamento, economizando esforço e preocupações concernentes a tal tarefa. Com uma arquitetura simples, de fácil configuração, e com funções de fácil entendimento, o Hibernate simplifica bastante a tarefa do desenvolvedor.

### Introdução a Estrutura de Dados: Tipos de Listas; Fila; Pilha; apresentar uma estrutura de dados implementada em JAVA (TED)

**Introdução a Estrutura de Dados**

Uma estrutura de dados é uma maneira de armazenar e relacionar conjuntos de informações de forma organizada e, na maioria das vezes, sequencial. Estas estruturas são muito importantes quando precisamos armazenar um conjunto de dados para ser utilizado em um determinado software. Na computação, há diversos tipos de estruturas de dados que podem ser utilizadas para diferentes fins.

A forma como estes dados serão agregados e organizados depende muito de como serão utilizados e processados, levando-se em consideração, por exemplo, a eficiência para buscas, o volume dos dados trabalhados, a complexidade da implementação e a forma como os dados se relacionam. Estas diversas formas de organização são as chamadas estruturas de dados.

**Tipos de Listas**

Uma Lista é uma estrutura de dados que é composta por nós, elementos, que apontam para o próximo elemento da lista, o último elemento apontará para nulo. Para compor uma lista encadeada, basta guardar seu primeiro elemento.

As listas podem ser alocadas de duas formas, sequenciais e encadeadas.

As listas sequenciais, também chamadas de arrays, arranjos ou vetores, são estruturas com posições fixas, onde cada elemento da lista deve ser colocado em uma posição no array. Ao utilizá-las para armazenamento de informações, é necessário definir em tempo de compilação qual o tamanho ou quantidade de bytes a ser alocados para o array.

As listas encadeadas não possuem restrição de tamanho, indicando que as células que as compõem devem estar alocadas sequencialmente na memória, consequentemente, não há definição de um tamanho fixo em tempo de compilação.

**Fila**

Uma fila é uma estrutura de dados dinâmica que admite remoção de elementos e inserção de novos objetos. Mais especificamente, uma fila é uma estrutura sujeita à seguinte regra de operação: sempre que houver uma remoção, o elemento removido é o que está na estrutura há mais tempo.

Em outras palavras, o primeiro objeto inserido na fila é também o primeiro a ser removido.

**Pilha**

Pilhas são estruturas de dados que armazenam os elementos em um formato sequencial, empilhando um item acima do outro. Estas estruturas permitem “empilhar” os itens que serão armazenados e “desempilhar” estes elementos da pilha quando for preciso removê-lo. Sempre que um novo elemento é inserido (ou empilhado) é dado a ele o nome de “topo”, pois é o primeiro elemento ao qual se terá acesso.

**Estrutura de dados implementada em JAVA**

A estrutura de dados escolhida para ser implementada foi do tipo pilha, o exemplo da implementação pode ser acessado através do link abaixo: <https://github.com/mthscoelho/wyden-desenv.software/tree/main/Tópico%2011%20-%20Pilha>

### Apresentar conceitos de estrutura de dados árvore AVL e um exemplo de implementação em JAVA

A AVL é uma árvore altamente balanceada, isto é, nas inserções e exclusões, procura-se executar uma rotina de balanceamento tal que as alturas das sub-árvores esquerda e sub-árvores direita tenham alturas bem próximas.

Uma árvore AVL é dita balanceada quando a diferença entre as alturas das sub-árvores não é maior do que um. Caso a árvore não estiver balanceada é necessário seu balanceamento através da rotação simples ou rotação dupla. O balanceamento é requerido para as operações de adição e exclusão de elementos. Para definir o balanceamento é utilizado um fator específico para nós.

**Implementação de Árvore AVL em Java**

O exemplo da implementação pode ser acessado através do link abaixo: <https://github.com/mthscoelho/wyden-desenv.software/tree/main/Tópico%2012%20-%20Árvore%20AVL>

### Introdução de estrutura de dados grafos e exemplos

Um grafo é uma estrutura de dados composta por um conjunto de vértices e um conjunto de arrestas. Sendo que cada arresta é representada por um par de vértices.

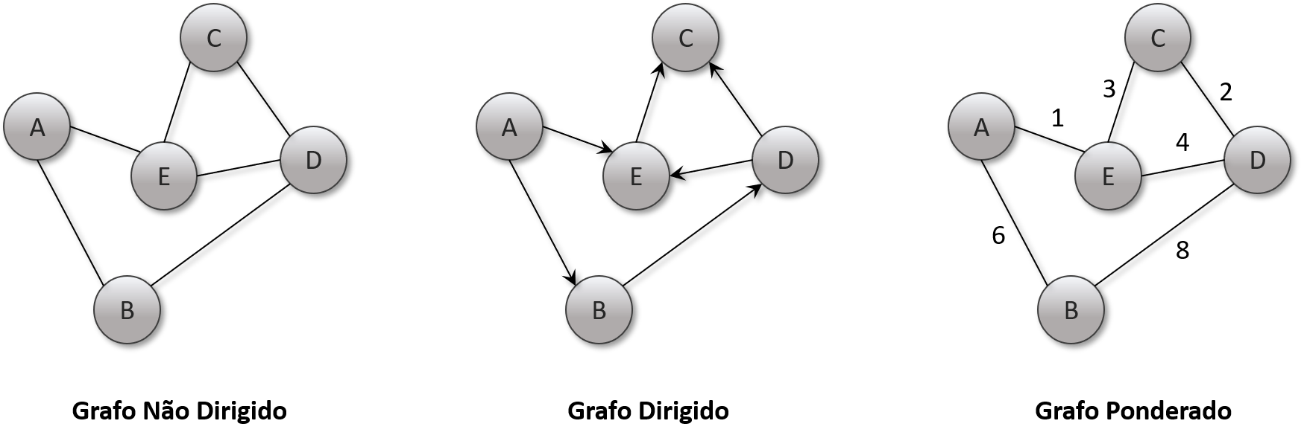
Grafos são bastante utilizados em redes sociais para manipular a grande quantidade de dados relacionados entre si que recebem a cada instante. Outra utilização de grafos, são em sistemas de navegação dos aplicativos de mapas/GPS, que utilizam grafos e o algoritmo de caminho mínimo para traçar rotas.

Os grafos podem ser direcionados e não direcionados.

• Grafo Não Dirigido: Um grafo não direcionado significa que, se tivermos uma aresta entre dois nós, A e B, então assumimos que existe um caminho de A para B, bem como de B para A.

• Grafo Dirigido: Um grafo direcionado tem setas nas arestas que representam a direção do relacionamento, ou seja, se tivermos uma aresta entre dois nós, A e B, direcionados de A para B, então o caminho existe apenas de A para B e não de B para A.

• Grafos Ponderados: Este tipo de grafo tem um peso ou número associado a cada borda.



### Descrição da problemática do caminho com custo mínimo; problema do caixeiro viajante, algoritmo Dijkstra

**Problemática do caminho com custo mínimo**

A problemática do caminho com custo mínimo ou caminho mais curto, consiste em encontrar o melhor caminho entre dois nós. Assim, resolver este problema pode significar determinar o caminho entre dois nós com o custo mínimo, ou com o menor tempo de viagem. Numa rede qualquer, dependendo das suas características, pode existir vários caminhos entre um par de nós, definidos como origem e destino. Entre os vários caminhos aquele que possui o menor “peso” é chamado de caminho mínimo. Este peso representa a soma total dos valores dos arcos que compõem o caminho e estes valores podem ser: o tempo de viagem, a distância percorrida ou um custo qualquer do arco.

**Problema do caixeiro viajante**

O Problema do Caixeiro Viajante é um problema que tenta definir a menor rota para percorrer uma série de cidades (visitando uma única vez cada uma delas), retornando à cidade de origem. Ele é um problema de otimização difícil inspirado na necessidade dos vendedores em realizar entregas em diversos locais (as cidades) percorrendo o menor caminho possível, reduzindo o tempo necessário para a viagem e os possíveis custos com transporte e combustível.

**Algoritmo de Dijkstra**

O Algoritmo de Dijkstra é um dos algoritmos que calcula o caminho de custo mínimo entre vértices de um grafo. Escolhido um vértice como raiz da busca, este algoritmo calcula o custo mínimo deste vértice para todos os demais vértices do grafo. Ele é bastante simples e com um bom nível de performance. Ele não garante, contudo, a exatidão da solução caso haja a presença de arcos com valores negativos.

Este algoritmo parte de uma estimativa inicial para o custo mínimo e vai sucessivamente ajustando esta estimativa. Ele considera que um vértice estará fechado quando já tiver sido obtido um caminho de custo mínimo do vértice tomado como raiz da busca até ele. Caso contrário ele dito estar aberto.

### Caracterização da linguagem JAVA: técnicas de programação (tipos primitivos; variáveis; constantes; operadores; estruturas de seleção; estruturas de repetição; arrays; functions)

**Caracterização da linguagem JAVA**

**Orientada a objetos**

Java é uma linguagem totalmente orientada a objetos, permitindo a herança e a reutilização de códigos de forma dinâmica e estática.

**Fácil e prática**

A linguagem é derivada da linguagem C e C++, tornando-se mais familiar. O ambiente retira do programador a responsabilidade de gerir a memória e os ponteiros. Esta linguagem incorpora também um suporte a operações em Banco de Dados, permitindo uma conexão remota a servidores SQL.

**Desempenho ao mais alto nível**

Suporta vários recursos de alto desempenho, como multithreading, compilação just-in-time e utilização de código nativo.

Excelente adaptabilidade para outras linguagens:

A máquina virtual Java roda cerca de 350 linguagens, o Groovy, Scala, JPython e outras.

**Portabilidade**

A linguagem JAVA permite desenvolver softwares que podem ser executados e distribuídos em diferentes plataformas (MAC, Linux, Windows etc.), sem a necessidade de modificá-los e, inclusive, sem pensar na arquitetura da máquina.

**Segurança**

A medida em que ganha escala, a internet necessita quase que de forma imediata de um ambiente seguro. Assim, partindo de tal necessidade e da ideia de que nada é confiável, o Java que segue permanecendo de acordo com este preceito. Buscou meios de manter seu compilador e o sistema em tempo de execução seguros da criação de códigos subversivos.

Esta é uma característica peculiar do Java. A plataforma está preparada para enfrentar os desafios de distribuir software dinamicamente pela rede. E isso, a partir da implementação de várias camadas de defesa contra códigos potencialmente incorretos.

**Técnicas de programação**

**Tipos primitivos**

Como tipos primitivos entendemos aqueles tipos de informação mais usuais e básicos. São os habituais de outras linguagens de programação.

Uma variável do tipo primitivo pode armazenar exatamente um valor de seu tipo declarado por vez, quando outro valor for atribuído a essa variável, seu valor inicial será substituído.

Os tipos primitivos são boolean, byte, char, short, int, long, float e double.

* Boolean: Não é um valor numérico, só admite os valores true ou false.
* Char: Usa o código UNICODE e ocupa cada caractere 16 bits.
* Inteiros: Diferem nas precisões e podem ser positivos ou negativos.

- Byte: 1 byte.

- Short: 2 bytes.

- Int: 4 bytes.

- Long: 8 bytes.

* Reais em ponto flutuante: igual que os inteiros também diferem nas precisões e podem ser positivos ou negativos.

- Float: 4 bytes.

- Double: 8 bytes.

As variáveis de instância de tipo primitivo são inicializadas por padrão, as variáveis dos tipos byte, char, short, int, long, float e double são inicializadas como 0, e as variáveis do tipo boolean são inicializadas como false. Esses tipos podem especificar seu próprio valor inicial para uma variável do tipo primitivo atribuindo à variável um valor na sua declaração.

**Variáveis**

Uma variável é uma estrutura que permite armazenar dados na memória durante a execução do programa, para processamento de informações.

Todas as variáveis devem ser declaradas antes que possam ser usadas. Declarar uma variável significa criá-la em algum ponto do programa

A linguagem Java é fortemente tipada. Isso significa que cada variável obrigatoriamente deve ter um tipo declarado antes que possa ser utilizada.

Em Java, todas as variáveis possuem um tipo e este será definido ainda na declaração. Pode-se alterar o valor armazenado numa variável a qualquer momento, porém, o tipo declarado não poderá ser alterado durante o tempo de execução. Mais precisamente, as variáveis são carregadas para a memória RAM devido à alta velocidade na leitura e gravação dessa mídia.

**Constantes**

Uma constante é um tipo especial de variável cujo valor, uma vez definido no código, não pode mais ser alterado durante a execução do programa.

Em programação, uma constante é a alocação de um espaço na memória para guardar um valor durante toda a execução de um programa ou então, durante a execução de um determinado Bloco de Instrução.

As constantes, são bastante semelhantes ao funcionamento e uso das variáveis, a diferença, é que não se pode alterar o seu valor em tempo de execução. Se, durante o desenvolvimento de um software, for definido que uma constante deve conter o valor inteiro 5, está, em hipótese alguma, conseguirá alterar o seu valor em tempo de execução.

**Operadores**

Operadores são símbolos que representam atribuições, cálculos e ordem dos dados. As operações seguem uma ordem de prioridades, ou seja, alguns cálculos (ou outros) são processados antes de outros.

Os operadores de atribuição, aritméticos, relacionais e lógicos no Java são utilizados principalmente na etapa de processamento, para a construção da lógica, possibilitando realizar ações específicas sobre os dados. Adição, subtração, multiplicação, comparação são apenas alguns exemplos.

**Operadores de atribuição em Java**

O operador de atribuição é utilizado para definir o valor inicial ou sobrescrever o valor de uma variável. Em seu uso, o operando à esquerda representa a variável para a qual se deseja atribuir o valor informado à direita.

**Operadores aritméticos**

Os operadores aritméticos realizam as operações fundamentais da matemática entre duas variáveis e retornam o resultado. Caso seja necessário escrever operações maiores ou mais complexas, pode-se combinar esses operadores e criar expressões, o que permite executar todo tipo de cálculo de forma programática.

**Operadores de incremento e decremento**

Os operadores de incremento e decremento também são bastante utilizados. Basicamente há dois deles: ++ e --, os quais podem ser declarados antes ou depois da variável e incrementam ou decrementam em 1 o valor da variável.

**Operadores de igualdade**

Os operadores de igualdade verificam se o valor ou o resultado da expressão lógica à esquerda é igual (“==”) ou diferente (“!=”) ao da direita, retornando um valor booleano.

**Operadores relacionais**

Os operadores relacionais, assim como os de igualdade, avaliam dois operandos. Neste caso, mais precisamente, definem se o operando à esquerda é menor, menor ou igual, maior ou maior ou igual ao da direita, retornando um valor booleano.

**Operadores lógicos**

Os operadores lógicos representam o recurso que permite criar expressões lógicas maiores a partir da junção de duas ou mais expressões. Para isso, é aplicado as operações lógicas E (representado por “&&”) e OU (representado por “||”).

**Estruturas de seleção**

A partir da verificação de uma condição, o programa realiza ou não uma ação e volta à sequência do programa. Existem quatro tipos:

1. Estruturas de Seleção Simples.

2. Estruturas de Seleção Compostas.

3. Estruturas de Seleção Encadeadas.

4. Estruturas de Seleção de Múltipla Escolha.

**Estruturas de Seleção Simples**

Esse tipo de estrutura verifica se a condição é verdadeira, caso seja, a instrução é executada, caso contrário, o fluxo do programa continua sem intervenção da estrutura de seleção.

**Estruturas de Seleção Compostas**

Esse tipo de estrutura possui dois conjuntos de instruções que verificam a condição, como resultado, caso seja verdadeira, realiza uma instrução e caso seja falsa, outra instrução é executada.

**Estruturas de Seleção Encadeadas**

Esse tipo de estrutura é uma sequência de instruções que podem ser ou não executadas dependendo condição validada em um grupo de opções específicas.

**Estrutura de Seleção Múltipla Escolha**

Este tipo de estrutura funciona como um conjunto de opções para escolha. Normalmente estas opções são pré-definidas e apresentadas em forma de tabela.

**Estruturas de repetição**

Estruturas de repetição, também conhecidas como loops (laços), são utilizadas para executar repetidamente uma instrução ou bloco de instrução enquanto determinada condição estiver sendo satisfeita.

As principais estruturas de repetição na maioria das linguagens são o for e o while.

**For**

O for é uma estrutura de repetição na qual seu ciclo será executado por um tempo ou condição pré-determinados e em uma quantidade de vezes que for determinado.

Quando utilizado o for, é preciso de uma variável para auxiliar a controlar a quantidade de repetições a serem executadas. Essa variável é chamada de variável de controle e é declarada no primeiro argumento do for.

**While**

O while também é uma estrutura de repetição, assim como o for. A diferença entre ambas é que, o for é utilizado quando geralmente é conhecida a quantidade de vezes que o trecho de código deverá ser repetido, é utilizado o while quando não se sabe exatamente quantas vezes o código será repetido.

**Arrays**

Arrays também são chamados de matrizes no Java, eles são como objetos que têm a característica de armazenar muitas variáveis de um mesmo tipo.

Dessa forma, eles são como objetos de recipientes. Assim, então, eles acabam por receber um número predeterminado de valores, que, novamente, devem ser de um mesmo tipo. Além disso, eles fazem parte da classe pertencente ao pacote java.util que está na API Java.

Os itens pertencentes a um Array são denominados elementos. Além disso, estes elementos são acessados a partir de um número, sendo este número chamado de índice. Este índice deve começar pelo 0.

**Funções**

Funções, também conhecidos como métodos, são responsáveis por ajudar no design do sistema (separação por blocos), e na reutilização de software, podendo ser aproveitados métodos já existentes para a construção de novos sistemas.

As classes que armazenam esses métodos são conhecidas como projetos de objetos, onde são definidos os atributos que o objeto terá e os métodos projetados para realizar as tarefas da classe.

**Métodos static em Java**

Os métodos static ou métodos da classe são funções que não dependem de nenhuma variável de instância, quando invocados executam uma função sem a dependência do conteúdo de um objeto ou a execução da instância de uma classe, conseguindo chamar direto qualquer método da classe e também manipulando alguns campos da classe.

Os métodos static tem um relacionamento com uma classe como um todo, enquanto os métodos que não são static são associados a uma instância de classe específica (objeto) e podem manipular as variáveis de instância do objeto.

**Método main**

Na execução de um programa Java, a JVM (Java Virtual Machine) tenta chamar o método main da classe que foi especificada. Quando declarado o método main como static permite que a JVM invoque o main sem criar uma instância da classe, ou seja, a classe é conhecida como classe principal ou classe testadora, que efetuará os testes e chamadas das classes para a execuções dos programas.

A JVM carrega a classe especificada pelo nome da classe que utiliza para invocar o método main (método principal/testador). Sempre que uma classe tiver esse método, é especificado que uma lista de Strings como argumentos de linha de comando, será passada para o aplicativo junto a JVM.

**Métodos sem retorno em Java**

Esse tipo de método executa apenas o código que tem dentro dele, não retornando nenhum resultado, sendo identificados com a palavra-chave void.

**Métodos com retorno em Java**

Esses métodos que não possuem a palavra-chave void incorporada na declaração, mas sim um tipo de dados, apresentam em seu corpo a palavra reservada return, que informa que o método terá que retornar o mesmo tipo de dados com o qual foi declarado.

### Apresentar um exemplo de backend em JAVA utilizando STS Spring Boot

Foi desenvolvido um backend em JAVA utilizando o framework Spring Boot.

Segue abaixo o link para o código hospedado no Github:

<https://github.com/mthscoelho/wyden-desenv.software/tree/main/Tópico%2016%20-%20Backend%20Spring%20Boot>

### Resolver 03 exercícios de uma das listas em JAVA: PUCRS ou UFBA

i. Escrever um algoritmo que lê 5 valores para a, um de cada vez, e conta quantos destes valores são negativos, escrevendo esta informação.

O algoritmo desenvolvido pode ser acessado através do link: <https://github.com/mthscoelho/wyden-desenv.software/tree/main/Tópico%2017%20-%20Exercício%2001>

ii. Escreva um algoritmo que leia o código de um aluno e suas três notas. Calcule a média ponderada do aluno, considerando que o peso para a maior nota seja 4 e para as duas restantes, 3. Mostre o código do aluno, suas três notas, a média calculada e uma mensagem "APROVADO" se a média for maior ou igual a 5 e "REPROVADO" se a média for menor que 5. Repita a operação até que o código lido seja negativo.

O algoritmo desenvolvido pode ser acessado através do link: <https://github.com/mthscoelho/wyden-desenv.software/tree/main/Tópico%2017%20-%20Exercício%2002>

iii. Chico tem 1,50 metro e cresce 2 centímetros por ano, enquanto Zé tem 1,10 metro e cresce 3 centímetros por ano. Construa um algoritmo que calcule e imprima quantos anos serão necessários para que Zé seja maior que Chico.

O algoritmo desenvolvido pode ser acessado através do link: <https://github.com/mthscoelho/wyden-desenv.software/tree/main/Tópico%2017%20-%20Exercício%2003>

1. CONSIDERAÇÕES FINAIS

Através desse trabalho, foi possível ampliar a compreensão acerca da área de desenvolvimento de software, linguagem JAVA, HTML, Spring Boot, UML, algoritmos e seus tópicos, também possibilitou a implementação de softwares, diagramas e algoritmos.

1. REFERÊNCIAS

AFONSO, A. Tutorial definitivo: Tudo o que você precisa para começar bem com JPA. Disponível em: <https://blog.algaworks.com/tutorial-jpa/>. Acesso em: 1 jun. 2022.

Desenvolvendo com Hibernate. Disponível em: <https://www.devmedia.com.br/desenvolvendo-com-hibernate/14756>. Acesso em: 6 jun. 2022.

IBM Docs. Disponível em: <https://www.ibm.com/docs/pt-br/was-zos/9.0.5?topic=SS7K4U\_9.0.5/com.ibm.websphere.base.doc/ae/cweb\_sov2.html>. Acesso em: 3 jun. 2022.

JUNIOR, E. JPA – Java Persistence API. O que é ? Como Funciona ? Disponível em: <https://www.tidicas.com.br/?p=1864>. Acesso em: 1 jun. 2022.

O Problema do Caixeiro Viajante. Disponível em: <http://www.mat.ufrgs.br/~portosil/caixeiro.html>. Acesso em: 7 jun. 2022.

O que é padrão MVC? Entenda arquitetura de softwares! Disponível em: <https://www.lewagon.com/pt-BR/blog/o-que-e-padrao-mvc>. Acesso em: 5 jun. 2022.

ORM: entenda o que é e saiba como usar no desenvolvimento. Disponível em: <https://blog.revelo.com.br/orm/>. Acesso em: 3 jun. 2022.

Padrões de projeto: o que são e o que resolvem. Disponível em: <https://www.treinaweb.com.br/blog/padroes-de-projeto-o-que-sao-e-o-que-resolvem>. Acesso em: 29 maio. 2022.

Por que escolher a ferramenta de diagramas UML do Lucidchart. , [s.d.]. Disponível em: <https://www.lucidchart.com/pages/pt/exemplos/uml-online>. Acesso em: 2 jun. 2022

ROVEDA, U. O que é programação orientada a objetos e quais são seus 4 pilares? Disponível em: <https://kenzie.com.br/blog/programacao-orientada-a-objetos/>. Acesso em: 5 jun. 2022.

WIKIPEDIA CONTRIBUTORS. Estrutura de dados. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Estrutura\_de\_dados&oldid=63377201>.

WIKIPEDIA CONTRIBUTORS. JavaServer Faces. Disponível em: <https://pt.wikipedia.org/w/index.php?title=JavaServer\_Faces&oldid=63631760>.