FACULDADE FACI WYDEN

José Felipe de Souza Lima

Tayane Caroliny Nunes Flexa Rodrigues

**DESENVOLVIMENTO DE SOFTWARE**

BELÉM – PA

2022

José Felipe de Souza Lima

Tayane Caroliny Nunes Flexa Rodrigues

**DESENVOLVIMENTO DE SOFTWARE**

O trabalho apresentado na disciplina Desenvolvimento de Software, como trabalho avaliativo, como requisito para obtenção da conclusão do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Heleno Filho

BELÉM – PA

2022

José Felipe de Souza Lima

Tayane Caroliny Nunes Flexa Rodrigues

**DESENVOLVIMENTO DE SOFTWARE**

**DESENVOLVIMENTO DE SOFTWARE**

O trabalho apresentado na disciplina Desenvolvimento de Software, como trabalho avaliativo, como requisito para obtenção da conclusão do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas..

Orientador: Prof. Iranildo Encarnação

Comissão Examinadora

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Prof. Heleno Filho

Faculdade Faci Wyden

BELÉM – PA

2022

SUMÁRIO

[1.ARQUITETURA WEB: FUNDAMENTOS; MODELOS ARQUITETURAIS 5](#_Toc105785056)

[2. HTML: EXEMPLO DE CONSTRUÇÃO DE CADASTRO DE FORMULÁRIO 6](#_Toc105785057)

[3. DESCREVER AS DIFERENÇAS ENTRE SERVLETS E SCRIPTS CGI 6](#_Toc105785058)

[4. INTRODUÇÃO AO JSP (JAVA SERVER PAGES) 8](#_Toc105785059)

[5. INTRODUÇÃO AOS PADRÕES DE PROJETOS; EXPLICAR FUNCIONAMENTO ARQUITETURA MVC 9](#_Toc105785060)

[6. INTRODUÇÃO AO ORIENTAÇÃO A OBJETOS 10](#_Toc105785061)

[7. INTRODUÇÃO A LINGUAGEM UML 12](#_Toc105785062)

[8. MODELAR UM DOMÍNIO 13](#_Toc105785063)

[9. INTRODUÇÃO A JSF (JAVASERVER FACES) 15](#_Toc105785064)

[10. APRESENTAÇÃO DOS CONCEITOS ORM E JPA E HIBERNATE 16](#_Toc105785065)

[11. INTRODUÇÃO A ESTRUTURA DE DADOS 16](#_Toc105785066)

[12. CONCEITOS DE ESTRUTURA DE DADOS 20](#_Toc105785067)

[13. INTRODUÇÃO DE ESTRUTURA DE DADOS GRAFOS E EXEMPLOS 23](#_Toc105785068)

[14. DESCRIÇÃO DA PROBLEMATICA DO CAMINHO COM CUSTO MÍNIMO, PROBLEMA DO CAXEIRO VIAJANTE, ALGORITMO DIJKSTRA 24](#_Toc105785069)

[15. CARACTERIZAÇÃO DA LINGUAGEM JAVA 26](#_Toc105785070)

[16. APRESENTAR UM EXEMPLO DE BACKEND EM JAVA UTILIZANDO STS SPRING BOOT 30](#_Toc105785071)

[17. RESOLVER OS 3 EXERCÍCIOS 33](#_Toc105785072)

[VÍDEO EXPLICANDO A TEMÁTICA 2 35](#_Toc105785073)

[REFERÊNCIAS BIBLIOGRÁFICAS 35](#_Toc105785074)

# 1.ARQUITETURA WEB: FUNDAMENTOS; MODELOS ARQUITETURAIS

Quando tenta-se solucionar um problema, é possível identificar diversas soluções que poderiam ser utilizadas visando resolvê-lo. Contudo, outros fatores como custo e eficiência influenciam na escolha da solução a ser adotada. No contexto do desenvolvimento de software, o mesmo pode ser observado ao se analisar os requisitos visando a construção de um software: várias soluções computacionais podem ser definidas para atender a esses requisitos, mas uma análise deve ser feita para definir a mais adequada ao contexto de desenvolvimento da aplicação. Para se representar essas soluções, a arquitetura de software é uma das abordagens que podem ser usadas.

Arquitetura de software representa a estrutura, ou conjunto de estruturas, que compreende os elementos de software, suas propriedades externamente visíveis e seus relacionamentos (BASS et al., 2003).

Para criar essa estrutura, grande parte dos autores concorda que três tipos de elementos básicos podem ser usados (DIAS e VIEIRA, 2000):

Elementos de software, podendo também ser chamados de módulos ou componentes, são as abstrações responsáveis por representar as entidades que implementam funcionalidades especificadas;

Conectores, podendo ser chamados de relacionamentos ou interfaces, são as abstrações responsáveis por representar as entidades que facilitam a comunicação entre os elementos de software;

Organização ou configuração que consiste na forma como os elementos de software e conectores estão organizados.

Os três padrões mais comuns são:

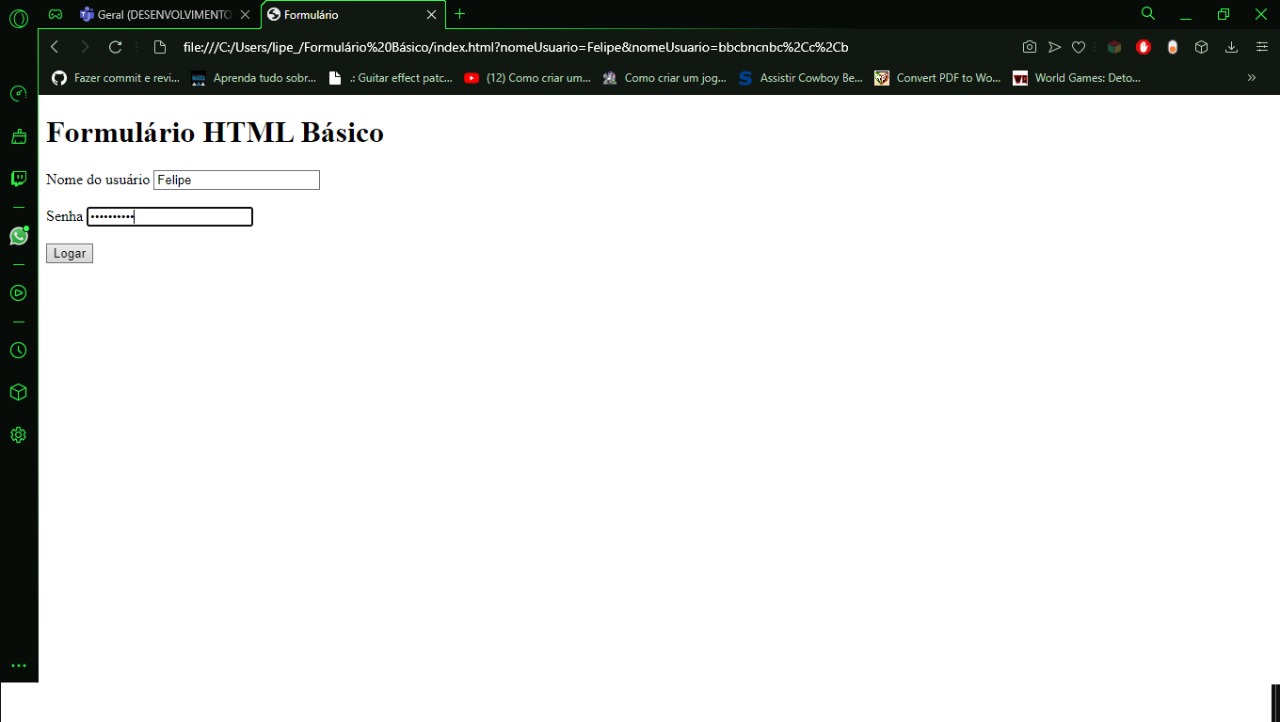
Cliente Web Thin - Utilizado principalmente para aplicativos com base na Internet, em que há pouco controle da configuração do cliente. O cliente requer apenas um navegador padrão da Web (com capacidade para formulários). Toda a lógica do negócio é executada no servidor.

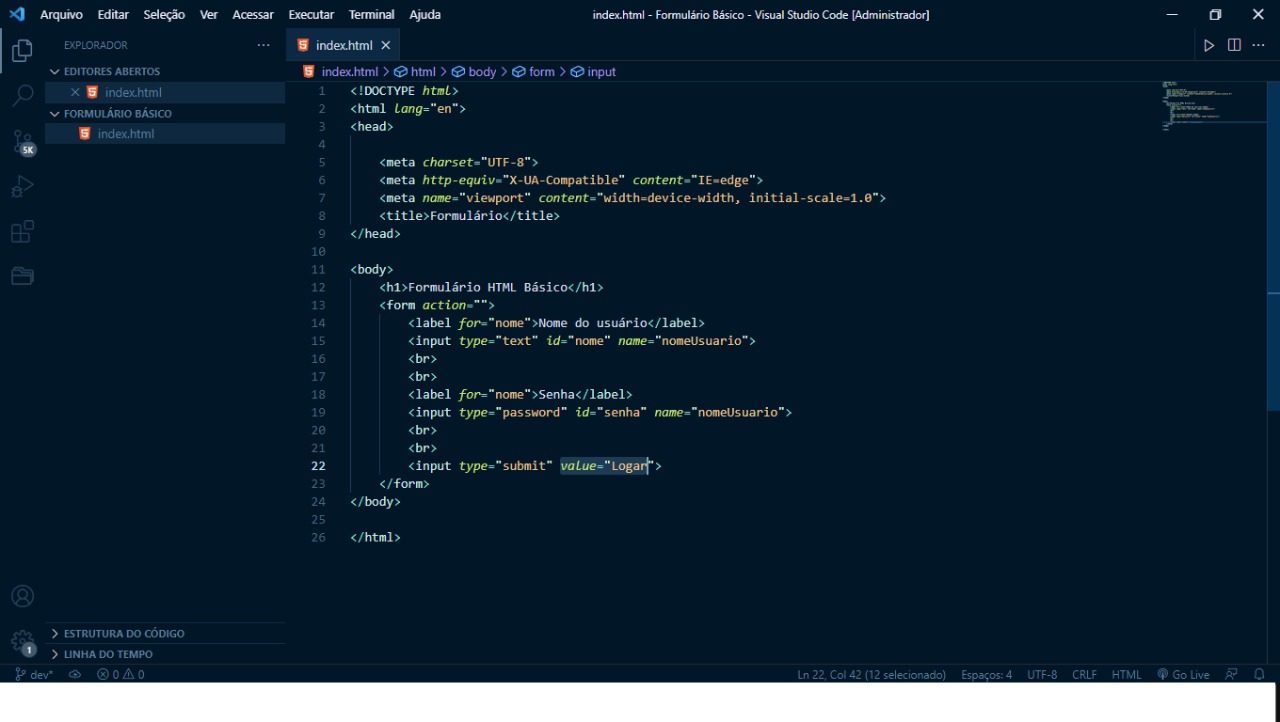
Cliente Thick Web - Uma parte significativa, em termos de arquitetura, da lógica de negócios é executada na máquina cliente. Em geral, o cliente utiliza HTML Dinâmico, Applets Java ou controles ActiveX para executar a lógica do negócio. A comunicação com o servidor ainda é feita via HTTP.

Entrega pela Web - Além de utilizar o protocolo HTTP para a comunicação de cliente e servidor, é possível utilizar outros protocolos, como IIOP e DCOM, para suportar um sistema de objetos distribuídos. O navegador da Web age principalmente como um dispositivo de entrega e de contêiner de um sistema de objetos distribuídos.

Essa lista não pode ser considerada completa, especialmente nos setores da indústria em que as revoluções tecnológicas parecem ocorrer a cada ano. Ela representa, em alto nível, os padrões mais comuns de arquitetura de aplicativos da Web. Assim como ocorre com qualquer padrão, é aceitável a aplicação de vários deles a uma única arquitetura.

# 2. HTML: EXEMPLO DE CONSTRUÇÃO DE CADASTRO DE FORMULÁRIO





# 3. DESCREVER AS DIFERENÇAS ENTRE SERVLETS E SCRIPTS CGI

O CGI (Common Gateway Interface) é uma interface que lida com programas externos (scripts CGI) em um servidor da Web para permitir a execução de páginas da Web interativas. O CGI foi desenvolvido pelo NCSA (Centro Nacional de Aplicações de Supercomputação) em 1993. Ele reside no lado do servidor e permite que os navegadores da Web interajam com os programas no servidor da web. Por exemplo, se uma página da Web consultar um banco de dados ou um usuário estiver enviando as informações do formulário para o servidor, os scripts CGI serão invocados. O servidor passa essas informações para um aplicativo de duas maneiras, GET ou POST, e o aplicativo responde ao servidor de volta ao navegador. Dessa maneira, os navegadores obtêm alguns resultados para o usuário.

O CGI é uma estipulação, para descrever um método para executar scripts CGI e fornecer os resultados de volta ao servidor para esses programas específicos. A função de um CGI é escanear as informações recebidas do navegador e produzir uma resposta apropriada, após a conclusão da tarefa o script CGI é finalizado. Na Common Gateway Interface, o comum significa que pode funcionar bem em qualquer sistema operacional ou linguagem de programação.

Um Servlet é um componente da Web baseado em Java que atua como um programa intermediário que facilita a interação entre o navegador da Web ou o cliente HTTP e o servidor HTTP. Semelhante ao CGI, os servlets também podem ser usados ​​para reunir as informações pelos formulários de páginas da Web, mostrar os registros do banco de dados e gerar páginas da Web dinâmicas com a ajuda de um contêiner. Servlet é uma classe Java que não depende da plataforma e ainda é compilada para o bytecode que é independente de plataforma. O bytecode neutro da plataforma pode ser armazenado e executado dinamicamente pelo servidor da Web Java. O servlet usa o mecanismo de soquete e RMI para estabelecer a conexão entre applets, bancos de dados ou outros programas de banco de dados.

Diferenças entre o CGI e o Servlet

Os scripts CGI são gravados no sistema operacional nativo e armazenados em determinado diretório. Por outro lado, os programas de servlet geralmente são escritos em Java, o qual é compilado no bytecode de Java e executado na JVM.

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.

Em CGI, cada solicitação de cliente que chega pode gerar um processo separado durante o servlet, os processos não são criados desnecessariamente e compartilham o espaço de memória da JVM.

Scripts CGI são programas executáveis ​​escritos no SO nativo do servidor. Por outro lado, os servlets são compilados para o bytecode Java que é executado na JVM.

O servlet é mais seguro que o CGI, pois usa o Java.

A velocidade, desempenho e eficiência do servlet é melhor que o CGI.

Scripts CGI podem ser processados ​​diretamente. Pelo contrário, o servlet primeiro traduz e compila o programa e o processa.

Quando se trata de portabilidade, o servlet é portátil, enquanto o CGI não é.

# 4. INTRODUÇÃO AO JSP (JAVA SERVER PAGES)

JSP é o acrônimo para Java Server Pages, uma linguagem criada pela SUN gratuita, JSP é uma linguagem de script com especificação aberta que tem como objetivo primário a geração de conteúdo dinâmico para páginas da Internet. Podemos ao invés de utilizar HTML para desenvolver páginas Web estáticas e sem funcionalidade, utilizar o JSP para criar dinamismo. É possível escrever HTML com códigos JSP embutidos. Como o HTML é uma linguagem estática, o JSP será o responsável por criar dinamismo. Por ser gratuita e possuir especificação aberta possui diversos servidores que suportam a linguagem, entre eles temos: Tomcat, GlassFish, JBoss, entre outros. O JSP necessita de servidor para funcionar por ser uma linguagem Server-side script, o usuário não consegue ver a codificação JSP, pois esta é convertida diretamente pelo servidor, sendo apresentado ao usuário apenas codificação HTML.

Uma página JSP possui extensão .jsp e consiste em uma página com codificação HTML e com codificação Java, inserida entre as tag´s <% e %>, denominada scriptlets e funcionando da seguinte forma: o servidor recebe uma requisição para uma página JSP, interpreta esta página gerando a codificação HTML e retorna ao cliente o resultado de sua solicitação. A página JSP que foi interpretada pelo servidor não precisa ser compilada como aconteceria com um servlet java por exemplo, esta tarefa é realizada em tempo real pelo servidor. É necessário apenas desenvolver as páginas JSP e disponibilizá-las no Servlet Container (Tomcat, por exemplo). O trabalho restante será realizado pelo servidor que faz a compilação em tempo de uso transformando o jsp em bytecode.

Assim, pode-se definir o 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.

Tecnologia Client-side e Server-side

A internet foi concebida de forma a funcionar como cliente X servidor, ou seja, temos um cliente que é o navegador web (browser) e o servidor http(web). Existem, na tecnologia disponível para Web, duas classificações de tecnologia, uma que funciona do lado do cliente ou Client-side e uma que funciona do lado do servidor ou Server-side.

Client-side – lado do cliente

O cliente-side de uma aplicação é o local onde ela é processada, ou seja, no caso da web, executa no navegador do cliente que é o responsável por interagir com o Servidor HTTP. Entre as tecnologias cliente-side temos o HTML que é executado no navegador, o CSS é outra tecnologia cliente que serve para formatar páginas HTML, há ainda o javascript que permite desenvolver ou ampliar o poder ao lado do cliente.

Server-side

Server-side, por sua vez, é o termo que representa o conjunto de tecnologias em que os processos são interpretados/processados diretamente no servidor, retornando como resultado a codificação client-side. Quando um cliente web (navegador) acessa uma página web, uma solicitação é enviada ao servidor através do protocolo http para que o servidor envie a resposta. O Servidor além de rodar os aplicativos, o lado servidor também é um repositório de páginas estáticas, que serão enviados ao cliente quando solicitado. Supondo que haja uma página JSP, esta será processada pelo servidor e encaminhado uma resposta ao cliente (Navegador).

# 5. INTRODUÇÃO AOS PADRÕES DE PROJETOS; EXPLICAR FUNCIONAMENTO ARQUITETURA MVC

O MVC é um padrão de arquitetura de software. O MVC sugere uma maneira para se pensar na divisão de responsabilidades, principalmente dentro de um software web.

O princípio básico do MVC é a divisão da aplicação em três camadas: a camada de interação do usuário (view), a camada de manipulação dos dados (model) e a camada de controle (controller).

Quando fala-se sobre o MVC, cada uma das camadas apresenta geralmente as seguintes responsabilidades:

Model: a responsabilidade dos models é representar o negócio. Também é responsável pelo acesso e manipulação dos dados na sua aplicação.

View: a view é responsável pela interface que será apresentada, mostrando as informações do model para o usuário.

Controller: é a camada de controle, responsável por ligar o model e a view, fazendo com que os models possam ser repassados para as views e vice-versa.

O MVC é de extrema importância para o desenvolvimento de software, uma dessas vantagens é a capacidade de tornar o código mais manutenível, ou seja, mais fácil de fazer manutenção, já que possui as responsabilidades devidamente separadas. Isso também traz uma facilidade na compreensão do código, além da sua reutilização. Além disso, o código se torna mais testável, pois ele é mais granular.

# 6. INTRODUÇÃO AO ORIENTAÇÃO A OBJETOS

Orientação a objetos é um paradigma aplicado na programação que consiste na interação entre diversas unidades chamadas de objetos. Pode ser definida por quatro pilares principais, sendo eles herança, encapsulamento, abstração e polimorfismo.

Classe e Objeto

Uma classe é uma forma de definir um tipo de dado em uma linguagem orientada a objeto. 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.

Atributos

Atributos de uma classe também conhecido como propriedades, descrevem um intervalo de valores que as instâncias da classe podem apresentar.

Um atributo é uma variável que pertence a um objeto. Os dados de um objeto são armazenados nos seus atributos. São informações do objeto, dados que pode armazenar.

Métodos

Os métodos são procedimentos ou funções que realizam as ações próprias do objeto. Assim, os métodos são as ações que o objeto pode realizar. Tudo o que o objeto faz é através de seus métodos, pois é através dos seus métodos que um objeto se manifesta, através deles que o objeto interage com os outros objetos.

Sendo mais conhecidos como: Método Construtor, Métodos Get e Set, Métodos do usuário e Método sobrescrito

Níveis de visibilidade

As visibilidade disponíveis são privativo (private), públicos(public) e protegidos(protect), eles definem como o atributo ou método se comporta dentro do meu programa.

Modificador Public

Quando você define um método ou atributo como público, você está liberando o acesso a ele de qualquer lugar que contenha a classe.

Modificador Protected

Esse modificador restringe o acesso de fora da classe, porém ele ainda é acessível de suas subclasses, ou seja, as classes que herdaram os atributos dela.

Modificador Private

Esse modificador é mais restritivo, só da permissão de acesso a sua própria classe, em nenhum outro lugar eu posso acessá-la.

Herança

A Herança possibilita que as classes compartilhem seus atributos, métodos e outros membros da classe entre si. Para a ligação entre as classes, a herança adota um relacionamento esquematizado hierarquicamente.

Na Herança temos dois tipos principais de classe:

Classe Base: A classe que concede as características a uma outra classe.

Classe Derivada: A classe que herda as características da classe base.

O fato de as classes derivadas herdarem atributos das classes bases assegura que programas orientados a objetos cresçam de forma linear e não geometricamente em complexidade. Cada nova classe derivada não possui interações imprevisíveis em relação ao restante do código do sistema.

Com o uso da herança, uma classe derivada geralmente é uma implementação especifica de um caso mais geral. A classe derivada deve apenas definir as características que a tornam única.

Encapsulamento

Encapsular os dados de uma aplicação significa evitar que estes sofram acessos indevidos. Para isso, é criada uma estrutura que contém métodos que podem ser utilizados por qualquer outra classe, sem causar inconsistências no desenvolvimento de um código.

Na prática, isso é feito por meio de dois métodos: os getters e os setters. O primeiro tem por objetivo retornar o valor que lhe foi pedido, mas de forma a não prejudicar a integridade do dado em si.

Já o segundo recebe como argumento uma informação, que pode ser qualquer tipo de dados suportados pela linguagem. Dessa forma, não haverá o risco de ocorrerem acessos indevidos.

Polimorfismo

O Polimorfismo é a possibilidade de em uma hierarquia de classes implementar métodos 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.

Extendeds e Implements

Implements é usado quando uma classe implementa uma interface, Extends é quando uma classe estende uma outra classe (concreta ou abstrata) ou quando uma interface estende uma outra interface. Ou seja, a keyword extends é usada para quando um tipo (classe ou interface) é derivado do seu mesmo tipo, e o implements é sempre que deseja fazer uma implementação, no caso uma classe implementa uma interface.

# 7. INTRODUÇÃO A LINGUAGEM UML

Devido ao grande avanço na utilização da orientação a objeto (OO) no desenvolvimento de sistemas, em 1994 dois conceituados metodologistas (James Rumbaugh e Grady Booch) efetuaram a unificação de seus métodos. Buscando minimizar os transtornos gerados pela ausência de Introdução a UML 73 minimizar os transtornos gerados pela ausência de uma metodologia padrão para OO. O método foi publicado sob o título Unified Method 0.8. No ano de 1995 outro metodologista, Ivar Jacobson, se juntou à dupla, gerando em outubro de 1996 a versão 0.91 da Unified Modeling Language ou apenas UML.

UML é um modeo de linguagem para modelagem de dados orientado a objeto, usada para especificar, construir, visualizar e documentar um sistema de software. Com ela podemos fazer uma modelagem visual de maneira que os relacionamentos entre os componentes do sistema sejam melhor visualizados e compreendidos e documentados. Tecnicamente dizendo, a UML (Unified Modeling Language) é a junção das três mais conceituadas linguagens de modelagem orientados a objetos (Booch de Grady, OOSE de Jacobson e o OMT de Rumbaugh).

Os elementos básicos de um diagrama de caso de uso são: ator, caso de uso, interação e sistema. Um ator é uma entidade externa ao sistema que de alguma forma participa de um caso de uso. Um ator pode ser um ser humano, máquinas, dispositivos, ou outros sistemas. Atores típicos são cliente, usuário, gerente, computador, impressora, etc. Os atores representam um papel e iniciam um caso de uso que após executado, retorna um valor para o ator. Um caso de uso especifica um serviço que será executado ao usuário e é composto por um ou mais cenários. Um cenário é uma narrativa de uma parte do comportamento global do sistema. Para melhor entendimento do Diagrama de Casos de Uso é necessária a descrição textual do fluxo do Caso de Uso (principal e alternativo) e do Cenário ou dos Cenários que compõe cada Caso de Uso.

# 8. MODELAR UM DOMÍNIO

Escolha da equipe: Sistema de locação de veículos.

Diagrama de casos de uso

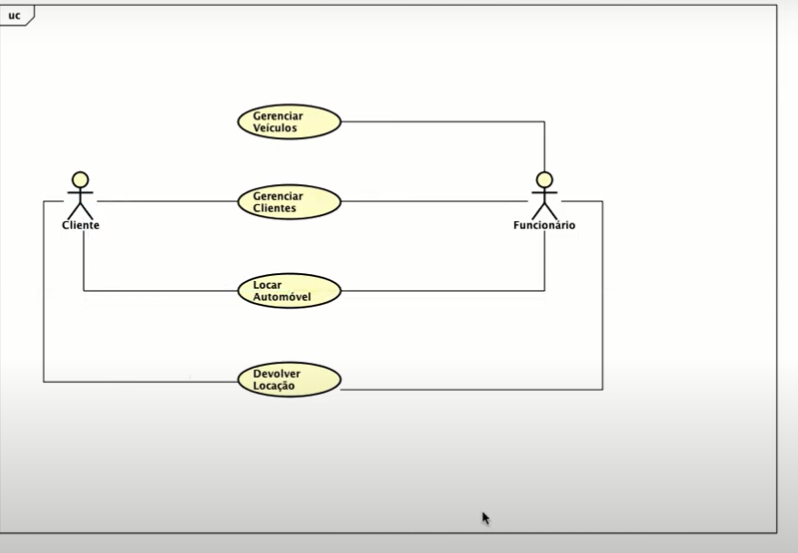


Diagrama de classe

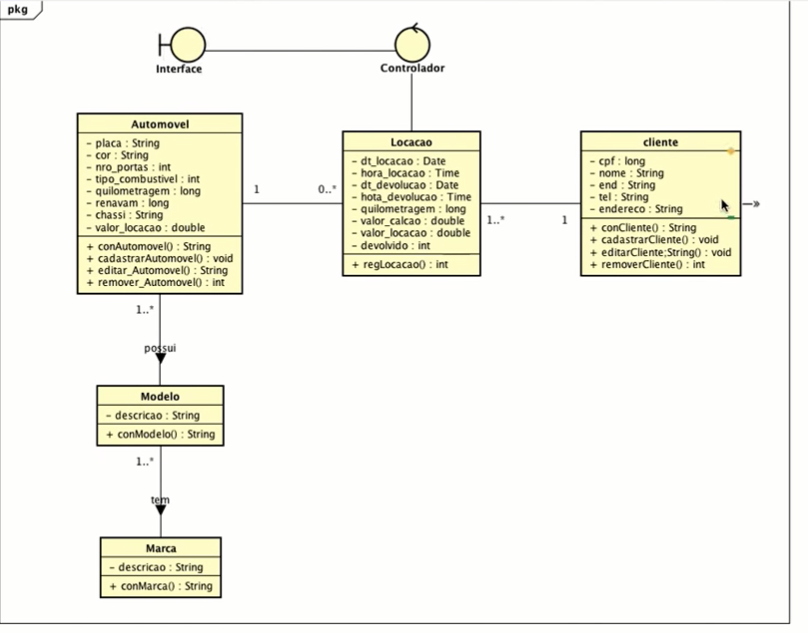
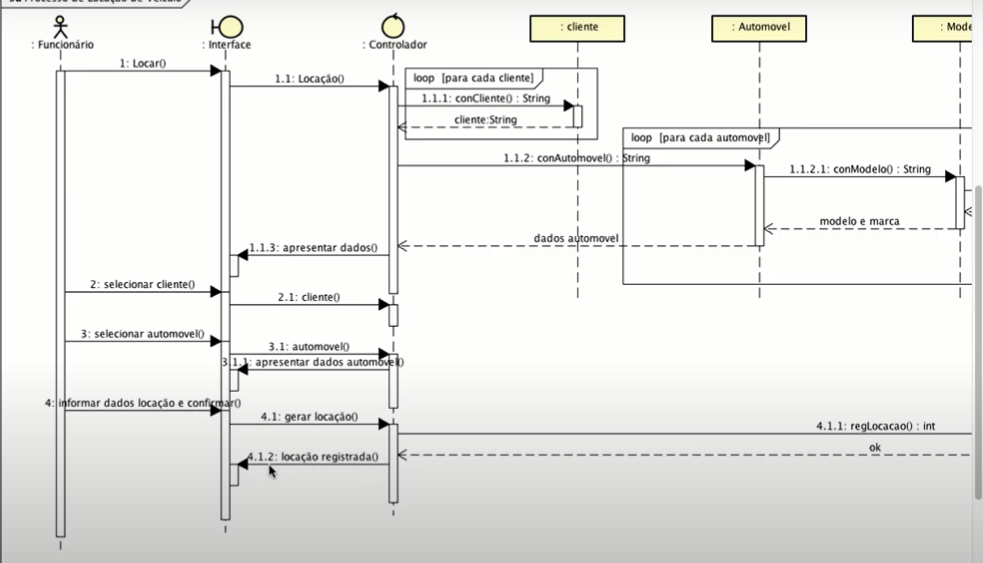
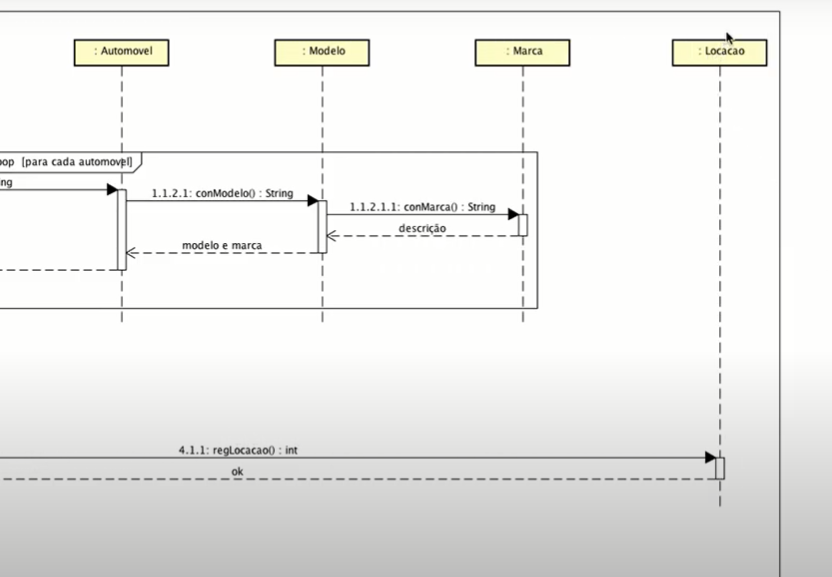


Diagrama de sequência





# 9. INTRODUÇÃO A JSF (JAVASERVER FACES)

JavaServer Faces é um framework MVC (model view controller – padrão de desenvolvimento que que separa as responsabilidades de um sistema em modelo, visão e controlador) que agiliza a construção de interfaces para a aplicação Java Web. Uma das características do JSF é trabalhar de uma maneira orientada a componentes de tela e seus eventos, por exemplo, cliques. Desta maneira, podemos associar estes componentes a diversos aspectos de nosso sistema, como a execução de operações de negócio, conversões de valores, validações de campos, etc. Este framework também suporta internacionalização, a utilização de layouts unificados, e [chamadas assíncronas ao servidor (AJAX)](https://www.devmedia.com.br/entendendo-o-ajax-em-4-passos/11997) -  é uma técnica que reúne um conjunto de tecnologias como DOM, XML e JavaScript, que permite a troca de informações de forma assíncrona com o servidor, tornando as páginas web muito mais dinâmicas, atraentes, flexíveis e robustas para os seus usuários.

Principais características: Tecnologia definida pelo JCP (Java Community Process), conjunto de componentes UI padrão, desenvolvedores podem criar componentes adicionais, componentes visuais se conectam com dados do servidor.

# 10. APRESENTAÇÃO DOS CONCEITOS ORM E JPA E HIBERNATE

ORM (Object Relational Mapper) é uma técnica de mapeamento objeto relacional que permite fazer uma relação dos objetos com os dados que os mesmos representam. Existem ótimos ORM´s como [Hibernate](https://www.devmedia.com.br/guia/hibernate/38312), [NHibernate](https://www.devmedia.com.br/introducao-ao-nhibernate-framework-para-mapeamento-objeto-relacional/28671), [Entity Framework](https://www.devmedia.com.br/entity-framework-tutorial/27764) e etc. Existem dois mundos: o relacional e o orientado a objetos.

No modelo relacional prevalecem princípios matemáticos com a finalidade de armazenar e gerenciar corretamente os dados, de forma segura e se trabalha com a linguagem SQL que é utilizada para dizer o banco de dados o que fazer e não como fazer.

E modelo orientado a objetos trabalhamos com classes e métodos, ou seja, trabalhamos fundamentados na engenharia de software e seus princípios que nos dizem como fazer. O ORM é a ponte entre estes dois modelos, ou seja, é ele quem vai permitir o armazenamento de objetos no banco de dados.

O ORM que faz o mapeamento da sua classe para o banco de dados e cada ORM tem suas particularidades para gerar o SQL referente a inserção do objeto que corresponde a uma tabela no banco de dados e realizar a operação. Utilizando um ORM, também se ganha produtividade, pois deixa-se de escrever os comando SQL para deixar que o próprio ORM, faça isto.

JPA ou Java Persistence API é uma camada que descreve uma interface comum para frameworks ORM, sendo a especificação padrão da plataforma Java EE (Enterprise Edition) para mapeamento objeto-relacional e persistência de dados. Para trabalhar com JPA é preciso incluir no projeto uma implementação, um exemplo seria o Hibernate.

Hibernate é um framework ORM, sendo a implementação física do que será usado para persistir, remover, atualizar ou buscar dados do banco. Ele que vai seguir as especificações do JPA no seu projeto, inclusive é responsável por todo trabalho de baixa serialização de dados como, por exemplo, a criação de uma query a ser consumida pelo JPA. Então, basicamente o Hibernate, seguindo as especificações do JPA, irá fazer a conversão de objetos em entidades baseadas em SQL, ou seja, ele faz todo o trabalho de baixo nível para converter dados entre paradigmas diferentes.

Hibernate é uma implementação e que o JPA é uma especificação. O JPA é apenas uma interface que define mapeamentos das classes e algumas configurações e o Hibernate apenas a classe que vai implementar essa interface.

# 11. INTRODUÇÃO A ESTRUTURA DE DADOS

Computadores são máquinas que manipulam dados e informações. A computação abrange o estudo da forma como as informações são organizadas, manipuladas e utilizadas em um computador.

Ao desenvolver um programa para realizar o processamento de dados, é preciso transcrever de forma que o computador possa compreender e executar tal programa e que o programador também compreenda o que escreveu. As linguagens de programação são códigos escritos em uma linguagem que o programador compreende e que o computador consegue interpretar e executar.

As estruturas de dados mais simples e que foram estudadas na disciplina de Programação Estruturada são os vetores e matrizes. Estas estruturas de dados são estruturas de dados homogêneos pois permitem o armazenamento de dados de um único tipo de dado. Elas permitem acesso direto a um elemento através do nome do vetor/matriz seguido do índice. Permitem também o acesso sequencial, percorrendo elemento a elemento do vetor/matriz.

Entretanto existem outras estruturas de dados mais complexas e que são usadas para representar de forma organizada arranjos de dados específicos. Existem as seguintes estruturas de dados:

Lista - 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

Filas - As filas são estruturas de dados baseadas na ideia de que o primeiro elemento a entrar na fila é o primeiro elemento a sair. Uma fila possui duas funções básicas que são adicionar elemento ao final da fila e remover o elemento que está no início da fila.

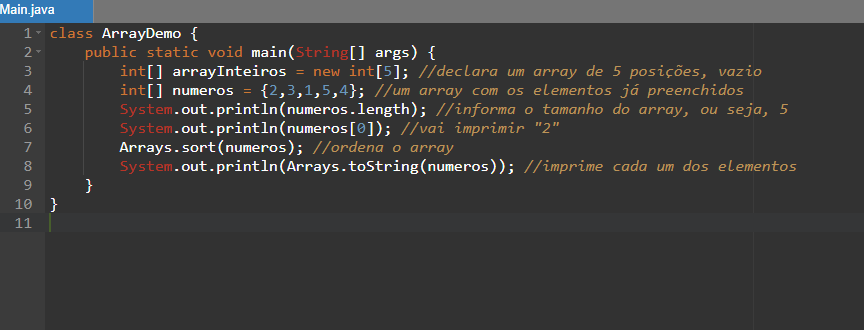
Pilha: São estruturas de dados do tipo LIFO (last-in first-out), onde o último elemento a ser inserido, será o primeiro a ser retirado. Assim, uma pilha permite acesso a apenas um item de dados - o último inserido. Para processar o penúltimo item inserido, deve-se remover o último.

Estrutura de dados em Java

Array

É uma estrutura linear onde dados homogêneos são armazenados em espaços contínuos da memória. O acesso a cada elemento é feito através de um índice. Como parte da modernização que Java vem passando, os organizadores da linguagem adicionaram ao Collections Framework classes utilitárias que seguem o esquema de nomenclatura: Array -> Arrays, Collection -> Collections, File -> Files.

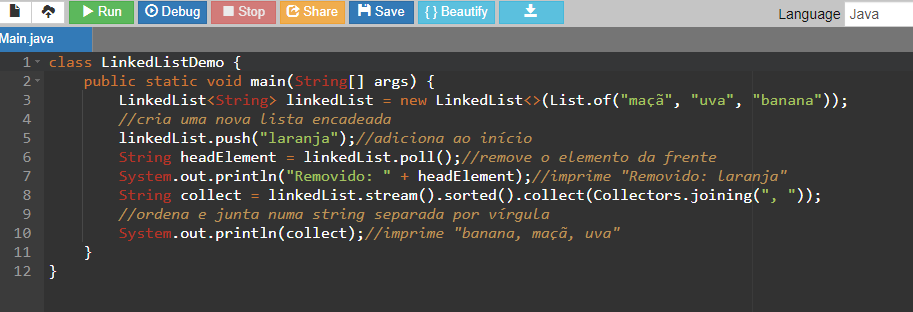
Exemplo de código:



Linked List

Existe um elemento "raiz", logo, cada elemento está ligado a outro. O tamanho é dinâmico e a lista pode ser "espalhada" pela memória.

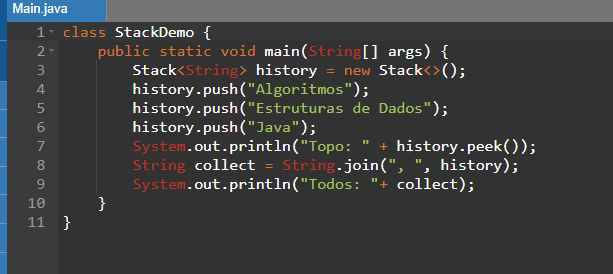
Essa estrutura costuma ser negligenciada devido suas semelhanças com ArrayList (a interface Java java.util.List implementada como um array). No entanto, a principal vantagem está em listas com tamanho dinâmico que costumam crescer quando necessário. Enquanto no melhor caso da operação de adição, ambas tem custo O(1).



Stack

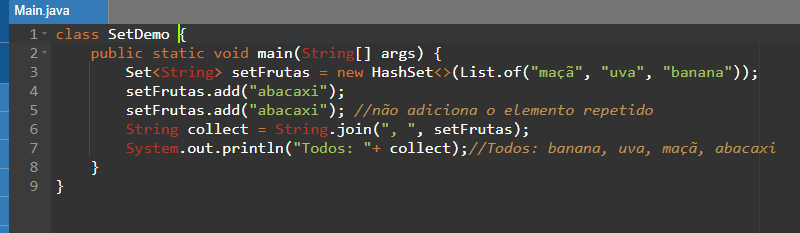
Representa uma pilha (como um baralho de cartas ou livros empilhados), seguindo o algoritmo LIFO. As quatro operações principais são: push(e) - insere um elemento no topo; pop() - remove o elemento do topo; isEmpty() - informa verdadeiro/falso se a pilha está vazia; peek() - acessa o elemento do topo (sem remover).

Essa estrutura é muito usada para representações de estado ou histórico de navegação, onde o mais recente está sempre no topo.



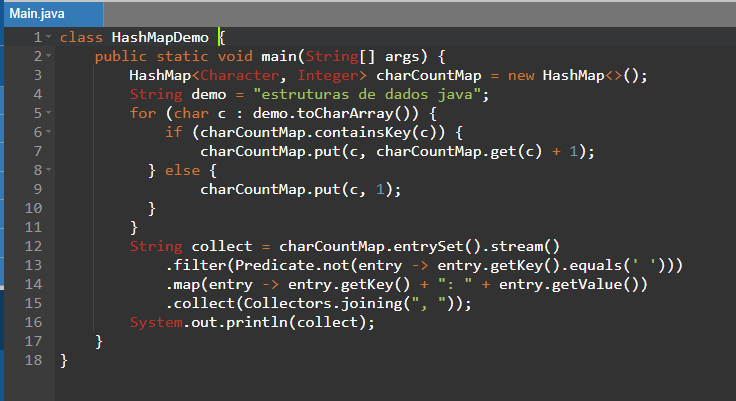
Set

Representa um grupo de dados homogêneos únicos, sem repetições. Por isso, deve haver uma forma de comparar os elementos de forma a decidir se são iguais ou não. Em Java, essa comparação é feita pelos métodos equals e hashcode. As principais classes que implementam a interface Set são: HashSet, TreeSet e LinkedHashSet.



HashMap

Usa funções hash para determinar a posição de cada elemento. Uma função hash é um algoritmo que mapeia dados de entrada de comprimento variável em dados de comprimento fixo. Assim, cada elemento a ser inserido na estrutura tem o seu endereço calculado. Isso evita "colisões" e acelera as operações de leitura.



# 12. CONCEITOS DE ESTRUTURA DE DADOS

As árvores binárias de pesquisa são projetadas para um acesso rápido à informação. Idealmente a árvore deve ser razoavelmente equilibrada e a sua altura será dada (no caso de estar completa) por h=log2 (n+1). O tempo de pesquisa tende a O(log2N) porém, com sucessivas inserções de dados principalmente ordenados, ela pode se degenerar para O(n)

Árvores completas são aquelas que minimizam o número de comparações efetuadas no pior caso para uma busca com chaves de probabilidades de ocorrências idênticas. Contudo, para garantir essa propriedade em aplicações dinâmicas, é preciso reconstruir a árvore para seu estado ideal a cada operação sobre seus nós (inclusão ou exclusão).

A AVL (Adelson-Velskii e Landis – 1962) 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, elas diferem no máximo por uma unidade.

Se o fator de balanceamento de qualquer nó ficar menor do que -1 ou maior do que 1 então a árvore tem que ser balanceada. Na rotação em AVL utiliza-se na inserção um processo de balanceamento que pode ser de 4 tipos específicos:

RR → caso Right-Right (rotação a esquerda)

LL → caso Left-Left (rotação a direita)

LR → caso Left-Right (rotação esquerda-direita)

RL → caso Right-Left (rotação direita-esquerda)

O fator de balanceamento é o Coeficiente que serve como referência para verificar se uma árvore AVL está ou não balanceada. O fator é calculado nó à nó e leva em consideração a diferença das alturas das sub-árvores da direita e da esquerda. Genericamente:

FB = he – hd

AVL em Java:

public class Arvore\_AVL {

No raiz;

public class No{

int info, fat\_bal;

No dir, esq;

public No(int dado){

esq = dir = null;

this.info = dado;

this.fat\_bal = 0;

}

public void inserir( int valor ){

if( valor < info ){

if( esq == null ){

esq = new No( valor );

}

else{

esq.inserir( valor );

}

}

else if( valor > info ){

if( dir == null ){

dir = new No( valor );

}

else{

dir.inserir( valor );

}

}

}

}

public Arvore\_AVL(){

{

raiz = null;

}

}

public void inserirNo( int valor ){

if( raiz == null )

{

raiz = new No( valor );

}

else

{

raiz.inserir( valor );

}

}

public void preOrdem()

{

preOrdemAjudante( raiz );

}

// Método para percorrer a árvore em pre - ordem

// utiliza a recursão para percorrer os nos da ávore

public void preOrdemAjudante( No no )

{

if( no == null )

return;

System.out.println( no.info + "" );

preOrdemAjudante( no.esq );

preOrdemAjudante( no.dir );

}

public static void main(String args[]){

Arvore\_AVL arvore = new Arvore\_AVL();

arvore.inserirNo(10);

arvore.inserirNo(20);

arvore.inserirNo(30);

arvore.preOrdem();

}

}

# 13. INTRODUÇÃO DE ESTRUTURA DE DADOS GRAFOS E EXEMPLOS

A estrutura de dados Grafos: Um Grafo é uma estrutura de dados formada por um conjunto de não vazio de vértices (ou nós) e por um conjunto de arestas (ou arcos), ligando estes vértices.

No grafo a seguir temos os seguintes conjuntos:

V = {A, B, C, D, E}

A = {(A,A), (A,B), (A, B), (A,C), (B,D), (B,E), (C,E)}

Seja G um grafo onde V é o conjunto dos vértices e A é o conjunto das arestas. Afirmar que A contém a aresta v-w, é o mesmo que afirmar que A contém a aresta w-v.

Definições importantes

1 - Um grafo é simples (ou regular) se não possuir laços e nem mais de uma aresta ligando dois vértices.

2 - A vizinhança de um nó é definida assim:

N(v) = {w pertence a V | v-w pertence a A}.

Nestes casos podemos dizer que o vértice w é adjacente a v e que a aresta v-w incide no vértice v.

3 - O grau de um vértice é a quantidade de arestas que incidem nele.

Um vértice é dito isolado se possuir grau zero.

4 - Um grafo completo com n vértices é um grafo simples onde existe uma aresta ligando todo par não ordenado vértices distintos.

O número máximo de arestas em um grafo com n vértices é n \* (n-1) /2.

5 - Um grafo não precisa ser uma árvore, mas toda árvore é um grafo.

# 14. DESCRIÇÃO DA PROBLEMATICA DO CAMINHO COM CUSTO MÍNIMO, PROBLEMA DO CAXEIRO VIAJANTE, ALGORITMO DIJKSTRA

O problema do caminho mais curto “Shortest path problem”, consiste em encontrar o melhor caminho entre dois pontos chamados nós. Assim, resolver este problema pode significar determinar o caminho entre dois nós com o custo mínimo, ou com o menor tempo de viagem ou com a máxima capacidade, (Ahuja et al., 1993).

O problema do caminho mais curto constitui o maior grupo na área de pesquisa operacional e é considerado como um dos mais importantes da programação linear (Hillier e Lieberman, 1988).

Glover et al. (1985), consideram o caminho mais curto como uma importante área de pesquisa devido à grande quantidade de aplicações práticas. As aplicações do problema do caminho mais curto estão relacionadas com a otimização de certas atividades como, por exemplo: o tráfego de estradas, linhas de transmissão elétrica, conexão de redes, problemas de programação de rota crítica PERT, planejamento de movimentos de um robô e outras mais complexas como no campo de biologia molecular, (Eppstein, 1994).

Ahuja et al. (1993), citam mais outras aplicações dos algoritmos para resolver o problema do caminho mais curto e estão relacionadas com a realização ótima de planos de substituição de equipamento, elaboração de projetos, gestão de fluxo de caixa, transmissão de mensagens em sistemas de comunicação e fluxo de tráfego em cidades com muita congestão.

Glover et al. (1985), mencionam as seguintes aplicações práticas: substituição de equipamento, planejamento de rotas e tempo de viagens, roteamento e programação de veículos, planejamento de capacidade, desenho e/ou expansão de redes de transporte e comunicação, e programação de caminhos críticos. Além das aplicações que estão associadas diretamente à busca do caminho mais curto, alguns outros problemas de programação inteira podem ser resolvidos através da formulação deste problema em forma direta ou aplicando alguma estratégia de relaxamento.

Glover et al. (1985) citam os seguintes problemas de programação inteira que foram resolvidos com o critério do caminho mais curto: localização, problema generalizado de atribuição, problema de emparelhamento máximo, problema do caixeiro viajante, problema da mochila e problemas de equilíbrio de tráfego.

Hung e Divoky (1990) mencionam que os algoritmos de busca do caminho mínimo têm sido utilizados para resolver problemas de emparelhamento, problemas de fluxo de custo mínimo e problemas de atribuição.

O problema do caminho mais curto é um tópico atrativo para pesquisadores e profissionais, pois sua resolução permite obter soluções eficientes a importantes e freqüentes problemas reais ao determinar de modo confiável a forma mais rápida e econômica de realizar uma atividade determinada. Outro ponto que faz com que o tema seja atrativo para pesquisadores é porque tem a característica de que a partir de modelos de redes simples podem-se elaborar modelos mais complexos que são estudados na área de otimização combinatória. Não obstante que a solução de problemas do caminho mais curto seja relativamente fácil, o desenho e a análise de algoritmos eficientes de solução exigem muito engenho, (Ahuja et al., 1993).

Problema do caixeiro viajante (ou Travelling Salesperson Problem, TSP) é um clássico exemplo de problema de otimização combinatória. A primeira coisa que pudesse pensar para resolver esse tipo de problema é reduzí-lo a um problema de enumeração: achando todas as rotas possíveis e, usando um computador, calculando o comprimento de cada uma delas e então vê-se qual a menor.

O algoritmo de Dijkstra encontra o menor caminho de um nó fonte ou origem s até os outros nós de uma rede orientada para o caso em que todos os pesos dos arcos sejam não negativos. Mantém a distância rotulada d(i) para cada nó i, e é o limite superior do caminho mais curto até o nó i. O algoritmo divide os nós em dois grupos: rotulados permanentemente (ou permanentes) e rotulados temporariamente (temporários). A distância rotulada dos nós permanentes representa a menor distância do nó fonte até um outro nó. No caso dos nós temporários, esta distância representa o limite superior da distância do caminho mais curto até o nó. O algoritmo examina todos os nós sem incluir o nó fonte s e rotula permanentemente os nós em ordem das distâncias ao nó fonte. Inicia-se o algoritmo com d(s) = 0 e , d i)( = ∞ para todo j que pertence à rede. Em cada iteração o rótulo do nó i é o menor caminho desde o nó fonte ao longo de um caminho que contém outros nós intermediários. O algoritmo escolhe o nó i com a mínima distância rotulada temporária para fazê-la permanente e visita os outros nós, ou seja, examina os arcos de A( i ) e atualiza as distâncias rotuladas dos nós adjacentes. O algoritmo termina quando todos os nós são designados como permanentes. A correção do algoritmo baseia-se na observação que é sempre possível designar o nó com a mínima distância temporária como permanente. O algoritmo de Dijkstra finaliza quando não existirem nós com rótulos temporários (caminho mais curto do nó s para todos os outros) ou quando o rótulo do nó i passar a permanente (caminho mais curto do nó s para o nó i).

# 15. CARACTERIZAÇÃO DA LINGUAGEM JAVA

O Java possui dois tipos de dados que são divididos por valor (tipos primitivos) e por referência (tipos por referência).

Os tipos primitivos são boolean, byte, char, short, int, long, float e double. Os tipos por referência, são classes que especificam os tipos de objeto Strings, Arrays Primitivos e Objetos.

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.

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.

O Java fornece dois tipos primitivos para armazenar números de ponto flutuante na memória, o tipo float e double.

A diferença entre eles é que as variáveis double podem armazenar números com maior magnitude e mais detalhes, ou seja, armazena mais dígitos à direita do ponto de fração decimal, do que as variáveis float. As variáveis do tipo float representam números de ponto flutuante de precisão simples e podem representar até 7 dígitos.

As variáveis do tipo double representam números de ponto flutuante de precisão dupla, onde precisam duas vezes a quantidade de memória das variáveis float fornecendo 15 dígitos, sendo o dobro da precisão de variáveis float. Os valores do tipo double são conhecidos como literais de ponto flutuante. Para números de ponto flutuante precisos, o Java fornece a classe BigDecimal (pacote java.math).

Uma variável é um recurso das linguagens de programação utilizado para armazenar valores em memória. Assim, sempre que precisarmos desse valor, basta referenciarmos essa variável. Em Java, podemos declarar variáveis, variáveis finais e constantes. As variáveis podem ter seu valor modificado a qualquer momento, enquanto as variáveis finais e constantes recebem um valor e não podem ser alteradas.

Na linguagem Java, declaramos uma variável informando o tipo de dados que ela poderá receber e seu nome.

O tipo de dado pode ser qualquer um dos tipos primitivos (como int, float, boolean), assim como qualquer outra classe/interface, seja ela nativa do Java (como String, ArrayList) ou criada por você ou por terceiros (como Produto, ProdutoDAO)

Nomeação de variáveis:

A nomeação de variáveis precisa ser feita respeitando algumas regras e convenções:

Pode conter letras, números e o caractere sublinhado (\_), mas não pode começar com um número;

Devem ser declaradas em minúsculo. Caso sejam nomes compostos, a primeira letra de todas as palavras, menos da primeira, deve ser maiúscula (Camel Case);

Java é uma linguagem case sensitive. Assim, numeroUm é diferente de numeroum.

Variáveis de classe

Ao declarar uma variável dentro de uma classe, antes do tipo também pode-se especificar o modificador de acesso.

A palavra private é um modificador de acesso. Em Java, também temos as opções public e protected.

Variáveis finais

Uma variável final é declarada quando precisa-se lidar com dados que não devem ser alterados durante a execução do programa. No Java declaramos uma variável final utilizando a palavra-chave final antes do tipo da variável. Abaixo um exemplo de declaração de uma variável final dentro do método main:

A variável final msgPadrao recebeu o valor "Olá!" e não pode ser alterado.

Uma observação quanto a criação de uma variável final, é que o seu valor não precisa ser atribuído no momento da sua criação. Podendo primeiro criar a variável final e atribuir o seu valor posteriormente.

Constantes

Assim como uma variável final, uma constante é declarada quando precisamos lidar com dados que não devem ser alterados durante a execução do programa. No Java declarando uma constante utilizando as palavras-chave static final antes do tipo da variável.

Uma observação é que uma constante é criada na classe, ou seja, fora do método. Por esse motivo utilizam o modificador de acesso 'public'.

Diferente de uma variável final, uma constante precisa receber o seu valor no momento em que ela for declarada.

Por convenção, usam-se letras maiúsculas para declarar constantes e assim distingui-las das variáveis.

Constantes são indicadas para substituir “números/valores mágicos”, isto é, aqueles números que aparecem no meio do código criado para realizar algum cálculo, agregando mais significado e facilitando a compreensão.

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 deseja-se 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, podem-se combinar esses operadores e criar expressões, o que permite executar todo tipo de cálculo de forma programática.

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, aplicam-se as operações lógicas E (representado por “&&”) e OU (representado por “||”).

Estrutura de seleção: A linguagem Java possui 3 estruturas de seleção, todas redutíveis a estrutura de seleção única básica da programação estruturada. São elas:

Estrutura if (seleção única): Se condição for verdadeira, o programa executa o bloco de instruções e continua para a próxima instrução.

Estrutura if/else (seleção dupla): Se condição for verdadeira, o programa executa o bloco de instruções 1, e se for falsa executa o bloco de instruções 2, após o que continua para a próxima instrução.

Estrutura switch (seleção múltipla): A estrutura verifica se o valor da variável critério é igual a cada uma das N possibilidades dadas por possibilidade1, possibilidade2, ... , possibilidadeN, e executa o bloco de instruções correspondente. Caso critério seja diferente de todas as possibilidades, o bloco de instruções alternativo é executado. A alternativa "default" é opcional. Após uma instrução "break" ou após executar a alternativa "default" o controle segue para a próxima instrução após a estrutura.

Estrutura de Repetição: Java implementa a estrutura de repetição de três formas diferentes:

Estrutura while: O programa verifica a condição e, se for verdadeira, executa a ação do bloco de instruções, após o que torna a verificar a condição. A execução do bloco de instruções é repetida até que a condição se torne falsa. Nesse caso, o controle do programa segue para a primeira instrução após a estrutura.

Estrutura do/while: Na estrutura do/while, o programa primeiro executa a ação definida pelo bloco de instruções e testa a condição. Se for verdadeira, o bloco de instruções é repetido e a condição é novamente testada. Se for falsa, o programa segue para a primeira instrução após a estrutura.

Estrutura for: Essa estrutura de repetição implementa um contador que irá controlar quantas vezes a ação será repetida. A (expressão1) define o contador e seu valor inicial, e a expressão3 incrementa ou decrementa o valor do contador cada vez que o bloco de instruções é executado. O loop é encerrado quando a condição definida em expressão2 deixar de ser verdadeira.

Arrays: ou matrizes, como são conhecidos pelo Java, fazem parte do pacote java.util na coleção da API do Java. São objetos de recipientes que contém um número fixo de valores de um único tipo. O comprimento de um array é estabelecido quando criado, sendo que após a criação o seu comprimento fica fixo.

Cada item em um array é chamado de elemento, e cada elemento é acessado pelo número, o índice. Sua numeração começa em 0.

Declaração de Arrays

Na declaração de um array, cada elemento recebe um valor padrão, sendo 0 (zero) para números de tipo primitivo, falso (false) para elementos booleanos e nulo (null) para referências.

Descobrindo o tamanho de um array

Por padrão, cada array sabe seu próprio tamanho, independente de quantos valores forem inseridos. O array armazena na variável de instância o método length, que retorna o tamanho do array especificado.

Percorrendo Arrays

A instrução for aprimorado, como é conhecida, serve para obter elementos de um array, sem a possibilidade de alterá-los. Se caso houver necessidade de modificar os elementos, isso pode ser feito com a instrução for controlada por contador.

Na sintaxe do for aprimorado é dividida pelos seguintes parâmetros:

O tipo com um identificador (parâmetro);

O nome do array no qual serão feitas as iterações.

Arrays multidimensionais

Esse tipo de array é declarado como tendo duas dimensões e é usado para representar tabelas de valores que consistem em informações organizadas em linhas e colunas.

Os arrays bidimensionais precisam de dois índices para identificar um elemento particular.

Por exemplo, quando um array é identificado dessa forma “numero[indiceA][indiceB]”, a variável número é o array, o indiceA é a linha e o indiceB é identificado como a coluna, fazendo uma identificação de cada elemento no array por número de linha e coluna.

# 16. APRESENTAR UM EXEMPLO DE BACKEND EM JAVA UTILIZANDO STS SPRING BOOT

O que é Spring Boot

O projeto foi criado pela [Spring](https://spring.io/) com o objetivo de facilitar o processo de configuração e publicação de aplicações, a fim de ter o projeto rodando o mais rápido possível e sem maiores dificuldades.

Isso é possível porque o Spring Boot consegue favorecer a convenção sobre a configuração. Ou seja, é só dizer quais módulos deseja utilizar (WEB, Template, Persistência, Segurança, etc.) que ele reconhece e faz as configurações.

Com a primeira versão, a ferramenta ficou conhecida principalmente pela facilidade de criação, configuração de projetos e a utilização de microsserviços. Além disso, o Spring Boot proporciona maior liberdade para pensar nas regras de negócio da aplicação.

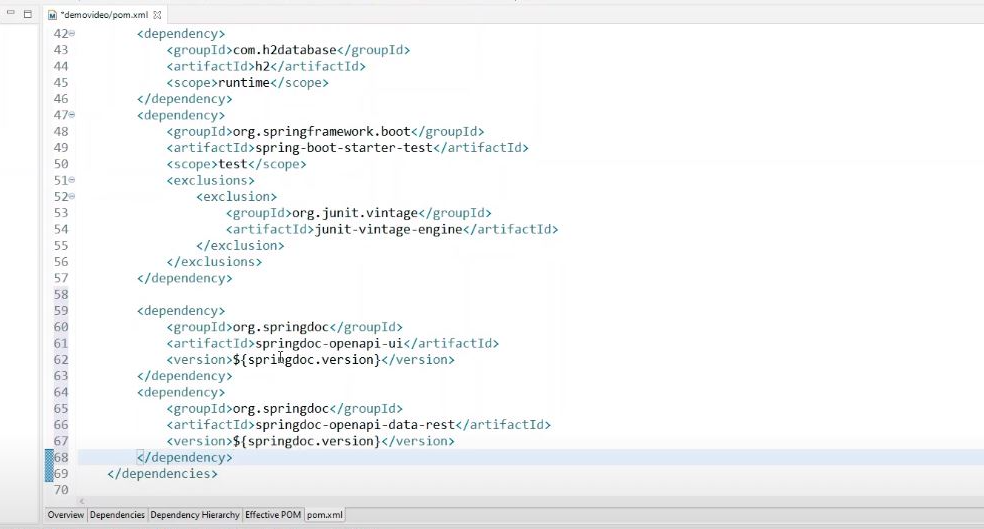
Backend API

Começando pelo significado, a sigla refere-se ao termo em inglês “Application Programming Interface” que traduzido para o português significa “Interface de Programação de Aplicativos”.

E essas interfaces são um conjunto de rotinas e padrões de programação para acesso a um aplicativo de software ou plataforma baseado na Web. Em resumo, uma API é composta por uma série de funções acessíveis somente por meio de programação.

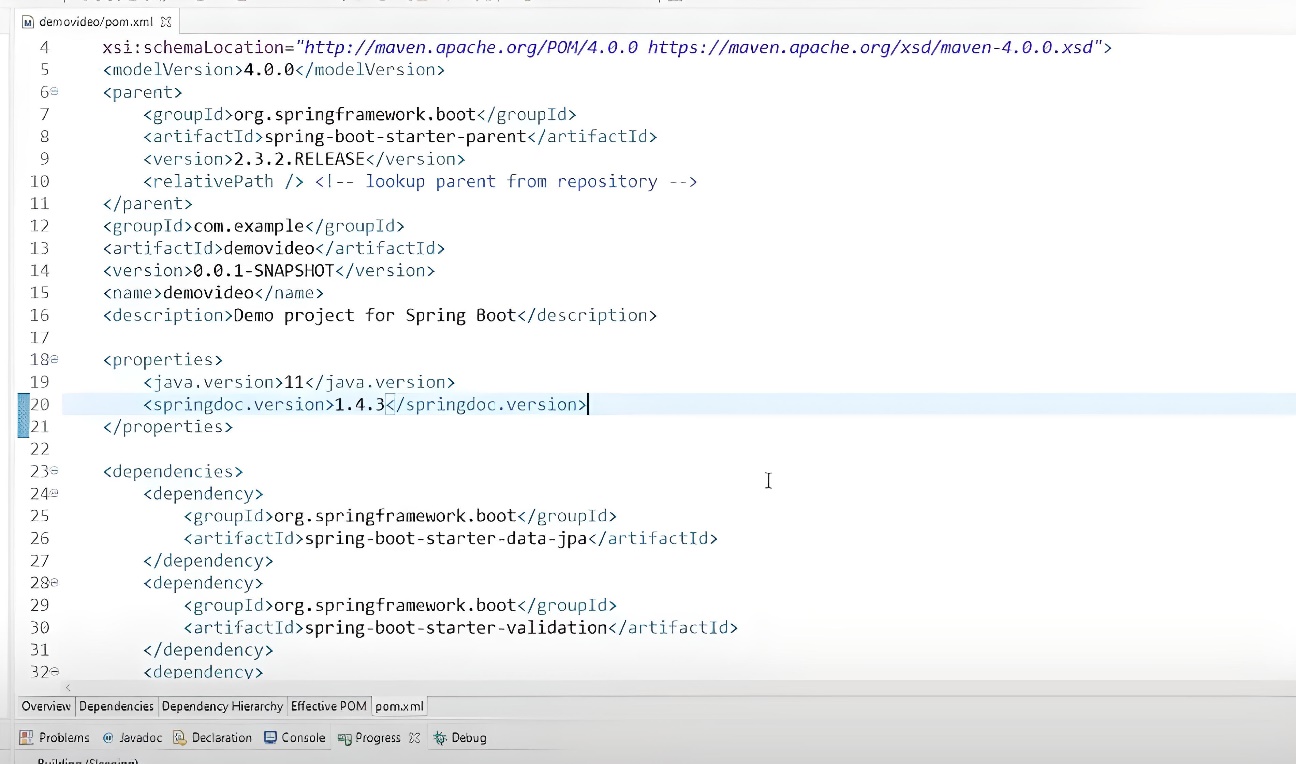
Mas afinal, quando APIs precisam ser criadas? Um caso comum é quando uma empresa de software tem intenção de que outros criadores de software desenvolvam produtos associados ao seu serviço.

Para exemplificar, o Google Maps é um dos mais utilizados. Por meio do código original, muitos outros sites e aplicações utilizam e adaptam os dados do Maps conforme seus objetivos.



Nas dependências do Swagger é necessário adicionar esse trecho no pom.xml para poder utilizar. Esse processo é manual por não ser uma dependência padrão da Spring, apesar de ser muito utilizada em aplicações Web.

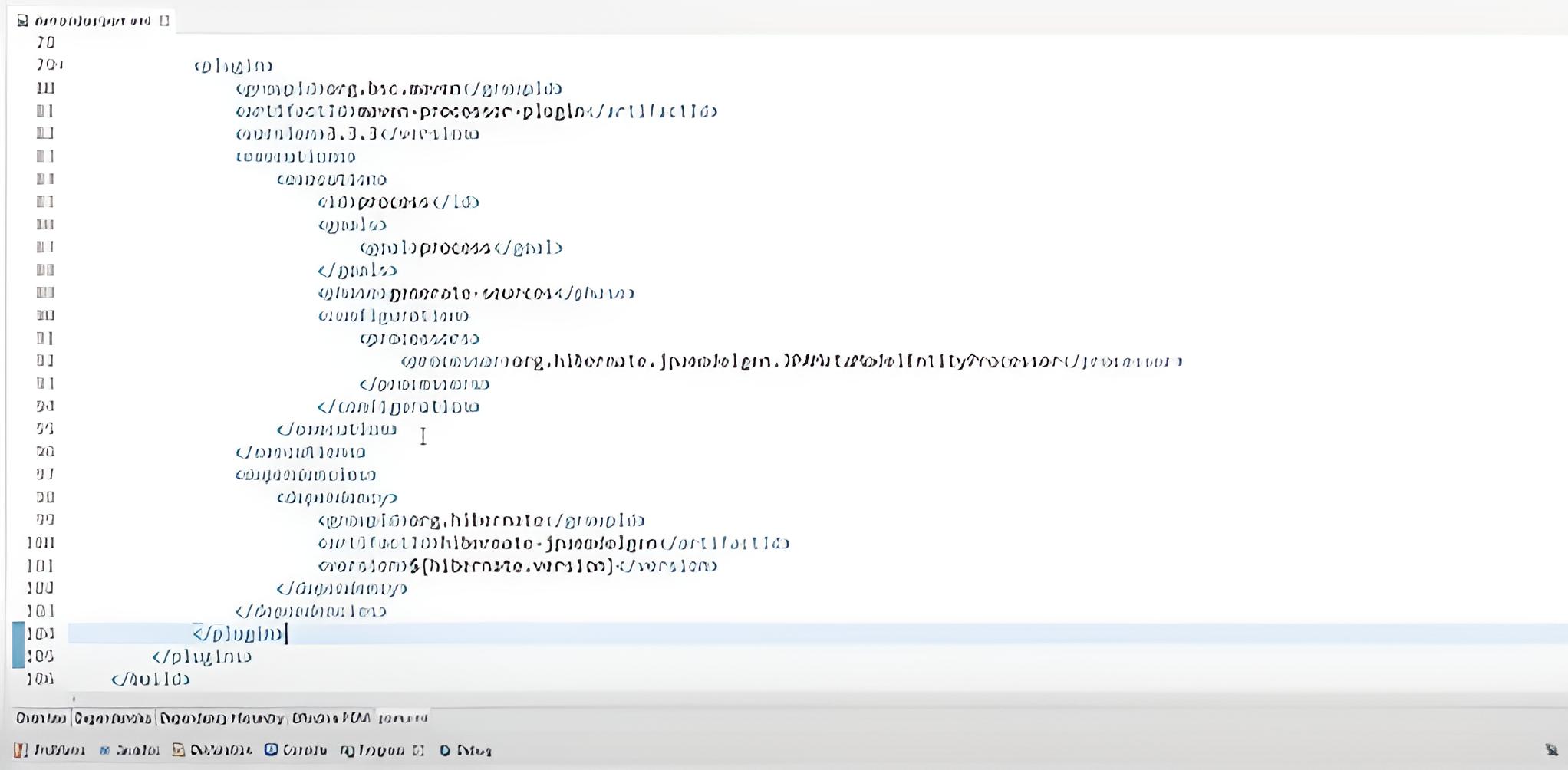
Properties



Os properties são propriedades internas do pom.xml, é onde se define as versões das dependências que são adicionadas manualmente no projeto.

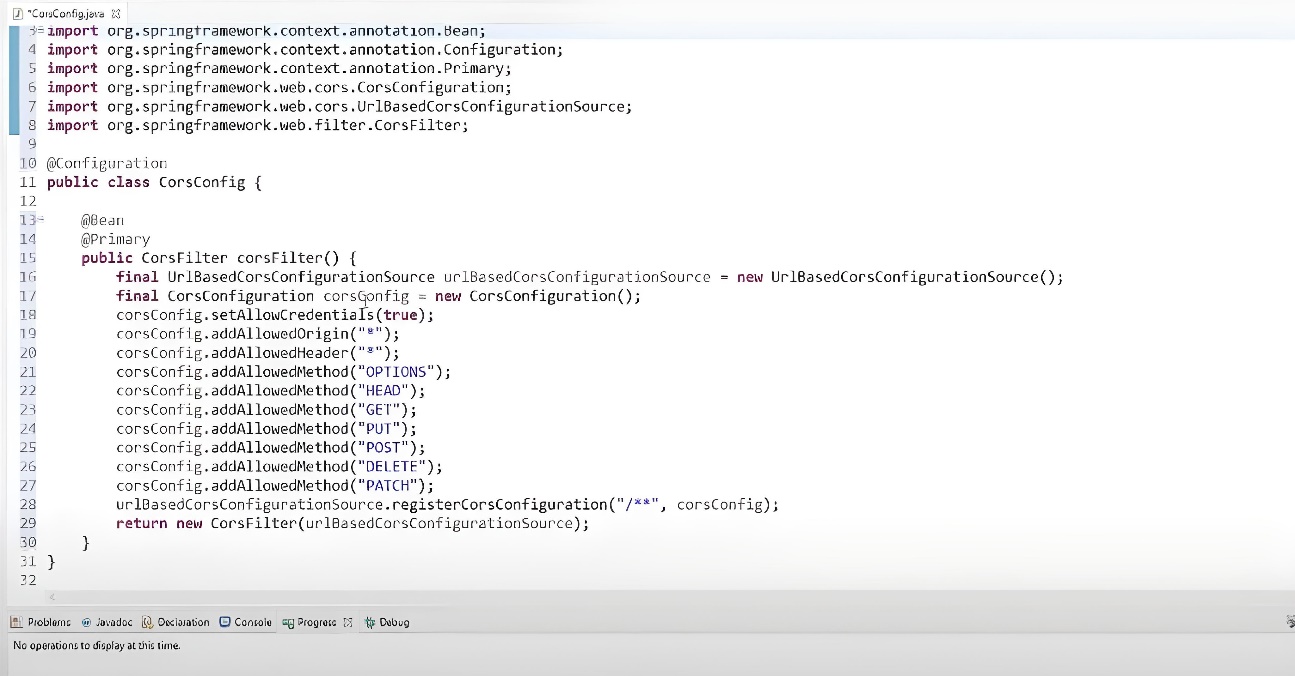
Essa config é opcional porque pode-se colocar o número da versão direto na tag <version></version> da dependência. As dependências do Spring (que foram selecionadas ao criar o projeto) não precisam estar ali, são inferidas automaticamente.

Metal model entity processor



O Meta Model Entity Processor é utilizado na criação das specifications para filtrar/pesquisar dados.

Configuração de cors



Cross-origin resource sharing é uma especificação de uma tecnologia de navegadores que define meios para um servidor permitir que seus recursos sejam acessados por uma página web de um domínio diferente. Esse tipo de acesso seria de outra forma negado pela same origin policy.

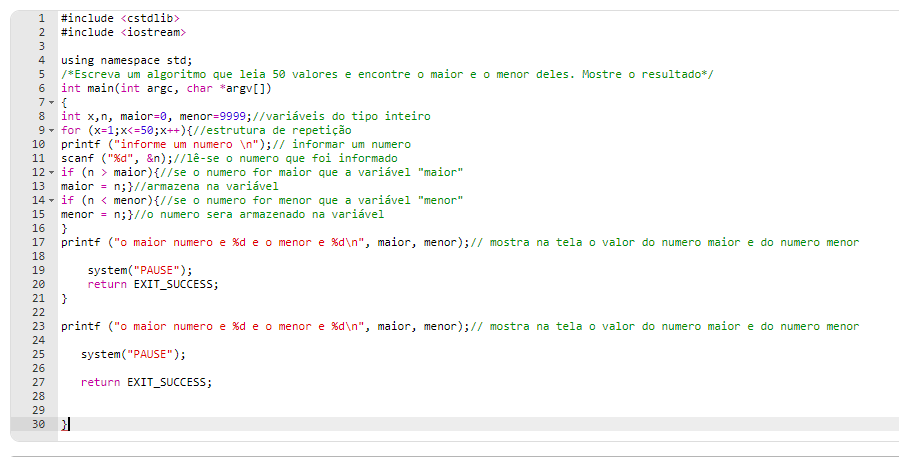
Essa configuração libera a API para ser consumida/utilizada publicamente por qualquer site ou origem, além disso não é necessário autorizar origem previamente.

# 17. RESOLVER OS 3 EXERCÍCIOS

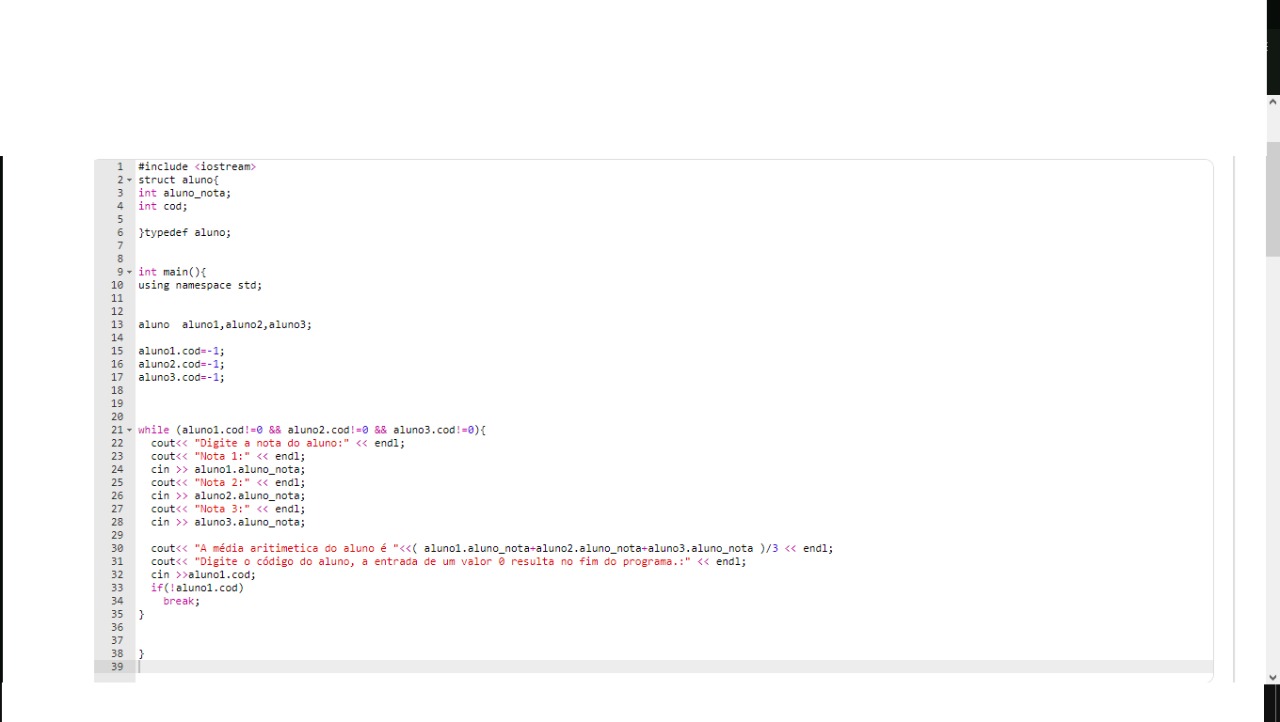
1 - Escrever um algoritmo que lê 5 valores para a, um de cada vez, e conta quantos destes valores são negativos, escrevendo esta informação.



2 - Escreva um algoritmo que leia 50 valores e encontre o maior e o menor deles. Mostre o resultado



3 - Escreva um algoritmo que calcule a média aritmética das 3 notas dos alunos de uma classe. O algoritmo deverá ler, além das notas, o código do aluno e deverá ser encerrado quando o código for igual a zero.



# VÍDEO EXPLICANDO A TEMÁTICA 2



# REFERÊNCIAS BIBLIOGRÁFICAS

Fundamentos da arquitetura de software. Disponível em: <https://www.devmedia.com.br/fundamentos-de-arquitetura-de-software/10581>.

Padrões de Arquitetura da Web. Disponível em: <https://www.cin.ufpe.br/~gta/rup-vc/core.base_rup/guidances/concepts/web_architecture_patterns_49E51CA1.html>.

Java Server Pages – JSP. Disponível em: <https://www.devmedia.com.br/introducao-ao-java-server-pages-jsp/25602#1>.

Tipos de dados por valor e por referência em Java. Disponível em: <https://www.devmedia.com.br/tipos-de-dados-por-valor-e-por-referencia-em-java/25293#:~:text=O%20Java%20possui%20dois%20tipos,Strings%2C%20Arrays%20Primitivos%20e%20Objetos>.

Java: variáveis e constantes. Disponível em: <https://www.devmedia.com.br/java-variaveis-e-constantes/38311#:~:text=Em%20Java%2C%20podemos%20declarar%20variáveis,variáveis%2C%20variáveis%20finais%20e%20constantes>.

Operadores de atribuição, aritméticos, relacionais e lógicos no Java. Disponível em: <https://www.devmedia.com.br/java-operadores-de-atribuicao-aritmeticos-relacionais-e-logicos/38289#:~:text=Os%20operadores%20de%20atribuição%2C%20aritméticos,comparação%20são%20apenas%20alguns%20exemplos>.

Operadores lógicos e matemáticos da linguagem Java. Disponível em: <https://www.devmedia.com.br/operadores-logicos-e-matematicos-da-linguagem-java/25248>.

As estruturas de controle de entrada/saída única de java. Disponível em: <http://www.ifba.edu.br/fisica/nfl/Java/linguagemestruturada.html#:~:text=SELEÇÃO%3A%20A%20linguagem%20Java%20possui,Estrutura%20if%20(seleção%20única).&text=Se%20condição%20for%20verdadeira%2C%20o,continua%20para%20a%20próxima%20instrução>.

O que são Arrays?. Disponível em: <https://www.devmedia.com.br/trabalhando-com-arrays-em-java/25530#:~:text=Os%20arrays%20ou%20matrizes%2C%20como,o%20seu%20comprimento%20fica%20fixo>.

Formulando o problema do caixeiro. Disponível em: <http://www.mat.ufrgs.br/~portosil/caixeiro.html#:~:text=O%20problema%20do%20caixeiro%20viajante%20consiste%20em%20descobrir%20a%20rota,e%20então%20volte%20a%20A>.

O que é MVC?. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-mvc>.

Orientação a objetos: Introdução e guia de estudos. Disponível em: <https://balta.io/blog/orientacao-a-objetos>.

O que é o encapsulamento?. Disponível em: <https://blog.betrybe.com/tecnologia/encapsulamento/#:~:text=O%20que%20é%20o%20encapsulamento,no%20desenvolvimento%20de%20um%20código>.

Principais conceitos da Programação Orientada a Objetos. Disponível em: <https://www.devmedia.com.br/principais-conceitos-da-programacao-orientada-a-objetos/32285#:~:text=Uma%20classe%20é%20uma%20forma,o%20comportamento%20são%20utilizados%20métodos>.

Conceitos e Exemplos. Disponível em: <https://www.devmedia.com.br/conceitos-e-exemplos-heranca-programacao-orientada-a-objetos-parte-1/18579>.

Conceitos de POO. Disponível em: <https://medium.com/@TDamiao/16-conceitos-poo-programação-orientada-a-objeto-6cdc72ac3ee2#:~:text=2%20Conceito%20—%20Atributos,Informações%20sobre%20o%20objeto>.

POO-Visibilidade de Atributos de métodos. Disponível em: <https://www.agenciadewebsites.com.br/?p=348#:~:text=As%20visibilidade%20disponíveis%20são%20privativo,comporta%20dentro%20do%20meu%20programa>.

Craindo api. Disponível em: <https://kbase.com.br/2020/09/16/criando-back-end-api-com-spring-boot-2-0/>.