**WYDEN**

**FACULDADE IDEAL**

**ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**GABRIEL LIMA RAMOS**

**WILDSON DA SILVA QUEIROZ**

**INTRODUÇÃO AO DESENVOLVIMENTO DE SOFTWARE**

**APRESENTAÇÃO DE CONCEITOS E IMPLEMENTAÇÕES**

**BELÉM-PA**

**2022**

**GABRIEL LIMA RAMOS**

**WILDSON DA SILVA QUEIROZ**

**INTRODUÇÃO AO DESENVOLVIMENTO DE SOFTWARE**

**APRESENTAÇÃO DE CONCEITOS E IMPLEMENTAÇÕES**

**BELÉM-PA**

**2022**

**RESUMO**

O presente trabalho, solicitado como critério de avaliação, buscou apresentar conceitos e a implementação de tópicos relacionados ao **desenvolvimento de software** definidos no contexto da disciplina Desenvolvimento de Software do curso tecnológico de Análise e Desenvolvimento de Sistemas (ADS) da Faculdade Ideal / Wyden.

**ABSTRACT**

The present work, requested as an evaluation criterion, sought to present concepts and the implementation of topics related to software development defined in the context of the Software Development discipline of the technological course of Systems Analysis and Development (ADS) at Faculdade Ideal / Wyden..

**SUMÁRIO**

[1. INTRODUÇÃO 5](#_Toc106364584)

[1.1. TÓPICOS 5](#_Toc106364585)

[1.1.1. Arquitetura Web: Fundamentos; Modelos Arquiteturais 5](#_Toc106364586)

[1.1.2. HTML: Exemplo de construção de cadastro de formulário 7](#_Toc106364587)

[1.1.3. Descrever as diferenças entre Servlets e Scripts CGI 8](#_Toc106364588)

[1.1.4. Introdução ao JSP (Java Server Pages) 10](#_Toc106364589)

[1.1.5. Introdução aos Padrões de Projetos; explicar funcionamento arquitetura MVC 10](#_Toc106364590)

[1.1.6. Introdução ao Orientação a Objetos: Classes; objeto; atributos/propriedades; métodos; níveis de visibilidade; herança; encapsulamento; polimorfismo; extends; implements 12](#_Toc106364591)

[1.1.7. Introdução a linguagem UML. 14](#_Toc106364592)

[1.1.8. Modelar um domínio (livre escolha), apresentar caso de uso, diagrama de classe e sequência 15](#_Toc106364593)

[1.1.9. Introdução a JSF (Java Server Faces) 18](#_Toc106364594)

[1.1.10. Apresentação dos conceitos ORM (Object Relacional Mapping) e JPA e Hibernate 18](#_Toc106364595)

[1.1.11. Introdução a Estrutura de Dados: Tipos de Listas; Fila; Pilha; apresentar uma estrutura de dados implementada em JAVA (TED) 19](#_Toc106364596)

[1.1.12. Apresentar conceitos de estrutura de dados árvore AVL e um exemplo de implementação em JAVA 20](#_Toc106364597)

[1.1.13. Introdução de estrutura de dados grafos e exemplos 26](#_Toc106364598)

[1.1.14. Descrição da problemática do caminho com custo mínimo; problema do caixeiro viajante, algoritmo Dijkstra 26](#_Toc106364599)

[1.1.15. Caracterização da linguagem JAVA: técnicas de programação (tipos primitivos; variáveis; constantes; operadores; estruturas de seleção; estruturas de repetição; arrays; functions) 28](#_Toc106364600)

[1.1.16. Apresentar um exemplo de backend em JAVA utilizando STS Spring Boot 30](#_Toc106364601)

[1.1.17. Resolver 03 exercícios de uma das listas em JAVA: PUCRS ou UFBA 31](#_Toc106364602)

1. INTRODUÇÃO

O desenvolvimento de software é o processo de conceber, especificar, projetar, programar, documentar, testar e corrigir bugs envolvidos na criação e manutenção de aplicativos, estruturas ou outros componentes de software.

O desenvolvimento de software envolve escrever e manter o código-fonte, mas em um sentido mais amplo, inclui todos os processos desde a concepção do software desejado até a manifestação final do software, normalmente em um processo planejado e estruturado.

O desenvolvimento de software também inclui pesquisa, novo desenvolvimento, prototipagem, modificação, reutilização, reengenharia, manutenção ou quaisquer outras atividades que resultem em produtos de software.

Desta forma, este trabalho buscou apresentar conceitos relacionados ao desenvolvimento de software da disciplina homônima e utilizando, principalmente, a linguagem Java para a implementação de alguns conceitos.

## TÓPICOS

Este trabalho é composto por **dezessete tópicos** e **um vídeo** apresentando um dos itens de livre escolha. Para o vídeo em questão, foi escolhido o **tópico 16** (apresentar exemplo de *back-end*).

O vídeo está disponível no endereço eletrônico abaixo:

<https://drive.google.com/file/d/18d6PdtI9pLjOMRGX-e5a38yuEEE0kKUF>

### Arquitetura Web: Fundamentos; Modelos Arquiteturais

A arquitetura web é composta de diversas camadas, conceitos e implementações. Desta forma, serão abordados abaixo os principais conceitos utilizados abaixo.

**Http**

“HTTP” significa Protocolo de Transferência de Hipertexto. Este é o protocolo usado para transferir documentos de hipertexto que tornam a World Wide Web possível. Quando utilizado em conjunto com o protocolo TSL/SSL, é denominado https.

A sua primeira versão foi implementada em 1989 por Tim Berners-Lee.

**Cliente/servidor**

Um dos principais padrões utilizados na web é a arquitetura cliente servidor. A arquitetura cliente servidor é uma arquitetura de aplicação distribuída, ou seja, na rede existem os fornecedores de recursos ou serviços a rede, que são chamados de servidores, e existem os requerentes dos recursos ou serviços, denominados clientes.

O cliente não compartilha nenhum de seus recursos com o servidor, mas no entanto ele solicita alguma função do servidor, sendo ele, o cliente, responsável por iniciar a comunicação com o servidor, enquanto o mesmo aguarda requisições de entrada.

**Website**

Um website ou apenas “site” é um conjunto de recursos Web relacionados, incluindo páginas Web, conteúdos multimédia, normalmente identificados com um nome de domínio comum e publicados em pelo menos um servidor Web. Exemplos notáveis ​​são wikipedia.org, google.com e amazon.com.

**Navegador**

Um navegador da Web (comumente chamado de navegador) é um agente de usuário de software para acessar informações na World Wide Web. Para se conectar ao servidor de um site e exibir suas páginas, o usuário precisa ter um programa de navegador da web. Este é o programa que o usuário executa para baixar, formatar e exibir uma página da Web no computador do usuário.

**Servidor**

A Web server is server software, or hardware dedicated to running said software, that can satisfy World Wide Web client requests. A web server can, in general, contain one or more websites. A web server processes incoming network requests over HTTP and several other related protocols.

The primary function of a web server is to store, process and deliver web pages to clients. The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP). Pages delivered are most frequently HTML documents, which may include images, style sheets and scripts in addition to the text content.

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

O formulário abaixo foi desenvolvido utilizando-se HTML, CSS, e JavaScript, bem como o framework css denominado Bootstrap (versão 5) e encontra-se disponível no endereço eletrônico: <https://queirozws.github.io/wyden-software-dev/>

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Figura 1 – Formulário desenvolvido utilizando HTML, CSS e JavaScript.

### Descrever as diferenças entre Servlets e Scripts CGI

O CGI e o Servlet são os programas que residem no servidor da web ou do aplicativo e auxiliam na comunicação entre o servidor da web e o navegador (lado do cliente) para gerar o conteúdo da web dinamicamente.

O CGI e o servlet 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.

**Definição de 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.

Anteriormente, os shell scripts do UNIX e o PERL eram usados para escrever os programas CGI, que é a razão pela qual eles são chamados de “scripts” CGI. Mas, agora, qualquer uma das linguagens como C, C ++, Perl, Visual Basic ou Python pode ser usada.

Embora o CGI possa liderar os possíveis problemas de segurança, como o servidor da Web também pode tratar arquivos executáveis como programas CGI em alguns diretórios específicos. O CGI manipula cada solicitação do cliente por um processo separado, o que aumenta a carga do servidor, tornando-a mais lenta.

**Definição de Servlet**

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. O contêiner de servlet é uma parte do servidor da Web que suporta os protocolos HTTP e HTTPS .

Ele emprega o método de solicitação / resposta usando o protocolo HTTP e HTTPS para permitir a interação com os clientes da web. Ao contrário do CGI, o servlet é executado dentro do espaço de endereço do servidor da web, onde cada cliente não é necessariamente tratado separadamente. No servlet, uma coleção de restrições é aplicada no servidor para proteger os recursos no servidor.

**Principais diferenças entre o CGI e o Servlet**

1. 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.

2. 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.

3. 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.

4. 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.

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

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

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

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

### Introdução ao JSP (Java Server Pages)

JavaServer Pages (JSP) é uma tecnologia padrão Java que permite escrever páginas dinâmicas e orientadas por dados para seus aplicativos da Web Java. JSP é construído sobre a especificação Java Servlet.

O objetivo da linguagem JSP não é só o desenvolvimento de páginas dinâmicas para Internet. Com ela é possível desenvolver sistemas inteiros para Internet. Além disso, existem diversos benefícios em se utilizar a linguagem JSP.

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.

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

MVC é conhecido como um padrão de arquitetura, que incorpora três partes: Model, View e Controller, ou para ser mais exato, divide a aplicação em três partes lógicas: a parte do modelo, a view e o controller. Foi usado para interfaces gráficas de usuário de desktop, mas hoje em dia é usado no design de aplicativos móveis e aplicativos da web.

**História**

Trygve Reenskaug inventou o MVC. Os primeiros relatórios sobre MVC foram escritos quando ele estava visitando um cientista no Laboratório de Pesquisa Xerox Palo Alto (PARC) em 1978/79. No início, o MVC era chamado de “Thing Model View Editor”, mas rapidamente o mudou para “Model View Controller”.

O objetivo do Tygrve era resolver o problema de usuários controlando um conjunto de dados grande e complexo. A prática do MVC mudou ao longo dos anos. Como o padrão MVC foi inventado antes dos navegadores da Web, inicialmente foi usado como um padrão de arquitetura para interfaces gráficas de usuário (GUI).

Diagrama

Descrição gerada automaticamente

Atualmente MVC é usado para projetar aplicações web. Alguns frameworks web que utilizam o conceito MVC: Ruby on Rails, Laravel, framework Zend, CherryPy, Symphony, etc.

**Arquitetura MVC**

MVC é um padrão arquitetural, o que significa que ele rege toda a arquitetura das aplicações. Embora muitas vezes seja conhecido como padrão de design, podemos estar errados se nos referirmos apenas como padrão de design, porque os padrões de design são usados ​​para resolver um problema técnico específico, enquanto o padrão de arquitetura é usado para resolver problemas de arquitetura, afetando todo o arquitetura do nosso aplicativo.

Tem três componentes principais:

- Model;

- View;

- Controller.

E cada um deles tem responsabilidades específicas

Diagrama

Descrição gerada automaticamente

As principais razões pelas quais o MVC é usado são: Primeiro, ele não permite que nos repitamos e segundo, ajuda a criar uma estrutura sólida de nossas aplicações web.

**Modelo**

É conhecido como o nível mais baixo, o que significa que é responsável pela manutenção dos dados. Manipule os dados logicamente para que basicamente lide com dados. O modelo está realmente conectado ao banco de dados, então tudo o que você faz com os dados. A adição ou recuperação de dados é feita no componente do modelo. Ele responde às solicitações do controlador porque o controlador nunca fala sozinho com o banco de dados. O modelo conversa com o banco de dados de um lado para o outro e, em seguida, fornece os dados necessários ao controlador. Nota: o modelo nunca se comunicou diretamente com a vista.

**Visão**

A representação dos dados é feita pelo componente view. Na verdade, ele gera UI ou interface de usuário para o usuário. Portanto, em aplicativos da Web, quando você pensa no componente de exibição, pense na parte Html/CSS. As visualizações são criadas pelos dados coletados pelo componente do modelo, mas esses dados não são obtidos diretamente, mas pelo controlador, portanto, a visualização fala apenas com o controlador.

**Controlador**

É conhecido como o homem principal porque o controlador é o componente que possibilita a interligação entre as visões e o modelo, por isso atua como um intermediário. O controlador não precisa se preocupar em lidar com a lógica de dados, apenas informa ao modelo o que fazer. Depois de receber os dados do modelo, ele os processa e, em seguida, pega todas essas informações, envia para a visualização e explica como representar para o usuário. Nota: Vistas e modelos não podem falar diretamente.

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

A programação orientada a objetos (OOP) é ​​um modelo de programação de computador que organiza o design de software em torno de dados ou objetos, em vez de funções e lógica. Um objeto pode ser definido como um campo de dados que possui atributos e comportamento exclusivos.

A estrutura, ou blocos de construção, da programação orientada a objetos incluem o seguinte:

As **classes** são tipos de dados definidos pelo usuário que atuam como o modelo para objetos, atributos e métodos individuais.

**Objetos** são instâncias de uma classe criada com dados especificamente definidos. Os objetos podem corresponder a objetos do mundo real ou a uma entidade abstrata. Quando a classe é definida inicialmente, a descrição é o único objeto definido.

Os **atributos** são definidos no modelo de classe e representam o estado de um objeto. Os objetos terão dados armazenados no campo de atributos. Os atributos de classe pertencem à própria classe.

**Métodos** são funções definidas dentro de uma classe que descrevem os comportamentos de um objeto. Cada método contido nas definições de classe começa com uma referência a um objeto de instância. Além disso, as sub-rotinas contidas em um objeto são chamadas de métodos de instância. Os programadores usam métodos para reutilização ou para manter a funcionalidade encapsulada dentro de um objeto por vez.

**Herança**. As classes podem reutilizar o código de outras classes. Relacionamentos e subclasses entre objetos podem ser atribuídos, permitindo que os desenvolvedores reutilizem a lógica comum enquanto ainda mantêm uma hierarquia única. Essa propriedade da OOP força uma análise de dados mais completa, reduz o tempo de desenvolvimento e garante um maior nível de precisão.

**Encapsulamento**. Este princípio afirma que todas as informações importantes estão contidas dentro de um objeto e apenas informações selecionadas são expostas. A implementação e o estado de cada objeto são mantidos em particular dentro de uma classe definida. Outros objetos não têm acesso a essa classe ou autoridade para fazer alterações. Eles só podem chamar uma lista de funções ou métodos públicos. Essa característica de ocultação de dados fornece maior segurança ao programa e evita corrupção de dados não intencional.

**Polimorfismo**. Os objetos são projetados para compartilhar comportamentos e podem assumir mais de uma forma. O programa determinará qual significado ou uso é necessário para cada execução desse objeto de uma classe pai, reduzindo a necessidade de duplicar o código. Uma classe filha é então criada, o que estende a funcionalidade da classe pai. O polimorfismo permite que diferentes tipos de objetos passem pela mesma interface.

A palavra ***Extends*** é usada para indicar que a classe que está sendo definida é derivada da classe base usando herança. Então, basicamente, a palavra-chave extends é usada para estender a funcionalidade da classe pai para a subclasse. Em Java, heranças múltiplas não são permitidas devido à ambiguidade. Portanto, uma classe pode estender apenas uma classe para evitar ambiguidade.

A palavra **implements** é usada para implementar uma interface. Uma interface é um tipo especial de classe que implementa uma abstração completa e contém apenas métodos abstratos. Para acessar os métodos da interface, a interface deve ser “implementada” por outra classe com a palavra-chave implements e os métodos precisam ser implementados na classe que está herdando as propriedades da interface. Como uma interface não possui a implementação dos métodos, uma classe pode implementar qualquer número de interfaces por vez.

### Introdução a linguagem UML.

Lançada no ano de 1997 (versão 1.0) pelo grupo denominado Object Management Group (OMG), a Unified Modeling Language (UML), buscou apresentar para a comunidade de software uma padronização para a modelagem de aplicações computacionais.

A UML não é uma metodologia, contudo, os seus diagramas facilitam o entendimento de uma aplicação.

Importante frisar que a UML foi elaborado considerando conceitos de Orientação a Objetos, tais como: classes, objetos, herança, abstração encapsulamento e polimorfismo.

Em sua versão 2.0, a UML possui 14 diagramas os quais podem ser agrupados em dois grandes grupos, a saber: diagramas estruturais e comportamentais. Atualmente está na versão 2.5.1.

Diagrama

Descrição gerada automaticamente

Figura 2 – Hieraquia dos diagramas UML 2.0.

**Diagramas estruturais**

Os diagramas estruturais são: Diagrama de classes, Diagrama de objetos, Diagrama de componentes, Diagrama de instalação ou de implantação, Diagrama de pacotes, Diagrama de estrutura composta, Diagrama de perfil.

**Diagramas comportamentais**

Os diagramas comportamentais são: Diagrama de caso de uso, Diagrama de transição de estados ou Máquina de estados, Diagrama de atividade, Diagrama de sequência, Diagrama Visão Geral de Interação ou de interação, Diagrama de colaboração ou comunicação, Diagrama de tempo ou temporal.

Cabe frisar que a literatura as vezes divide os diagramas comportamentais acima em: diagramas comportamentais e de interação.

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

O domínio escolhido foi o de Ordem de Serviço (Aplicação de pedido de comida. Ex: iFood) e abaixo são apresentados os diagramas de classes, casos de uso e sequência para o domínio em questão.

**i. Diagrama de classes**

É apresentado abaixo o diagrama de classe desenvolvido para a aplicação.

Figura 3 – Diagrama de classe.

**ii. Diagrama de casos de uso**

É apresentado abaixo o diagrama de sequência desenvolvido para a aplicação.

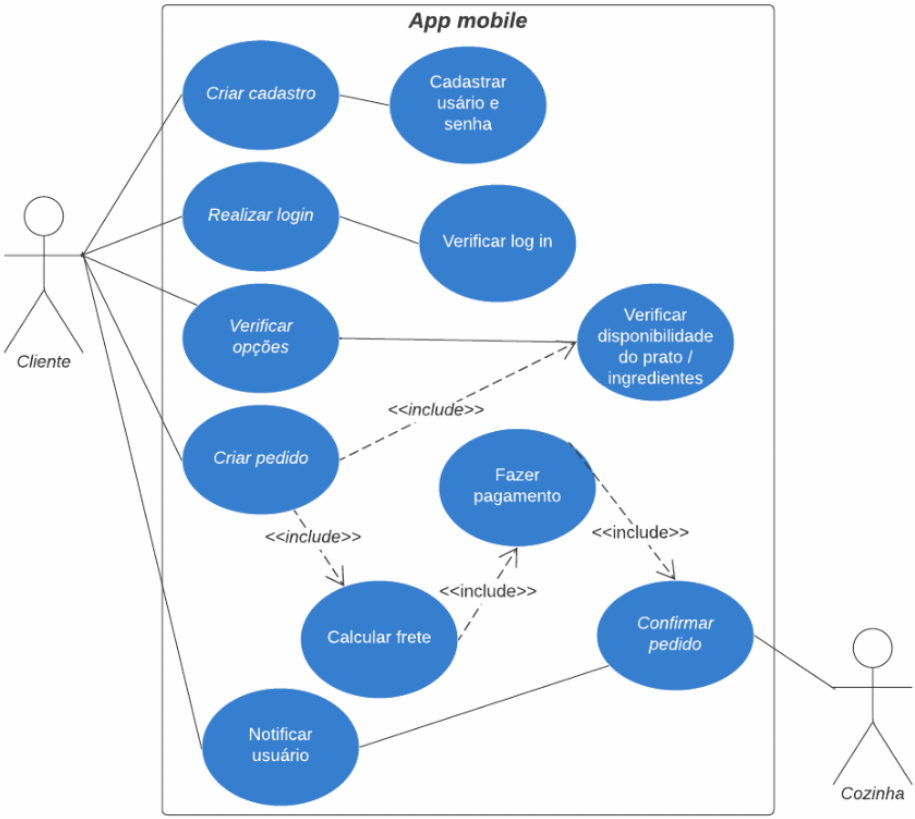


Figura 4 – Diagrama de casos de uso.

**iii. Diagrama de sequência**

É apresentado abaixo o diagrama de sequência desenvolvido para a aplicação.

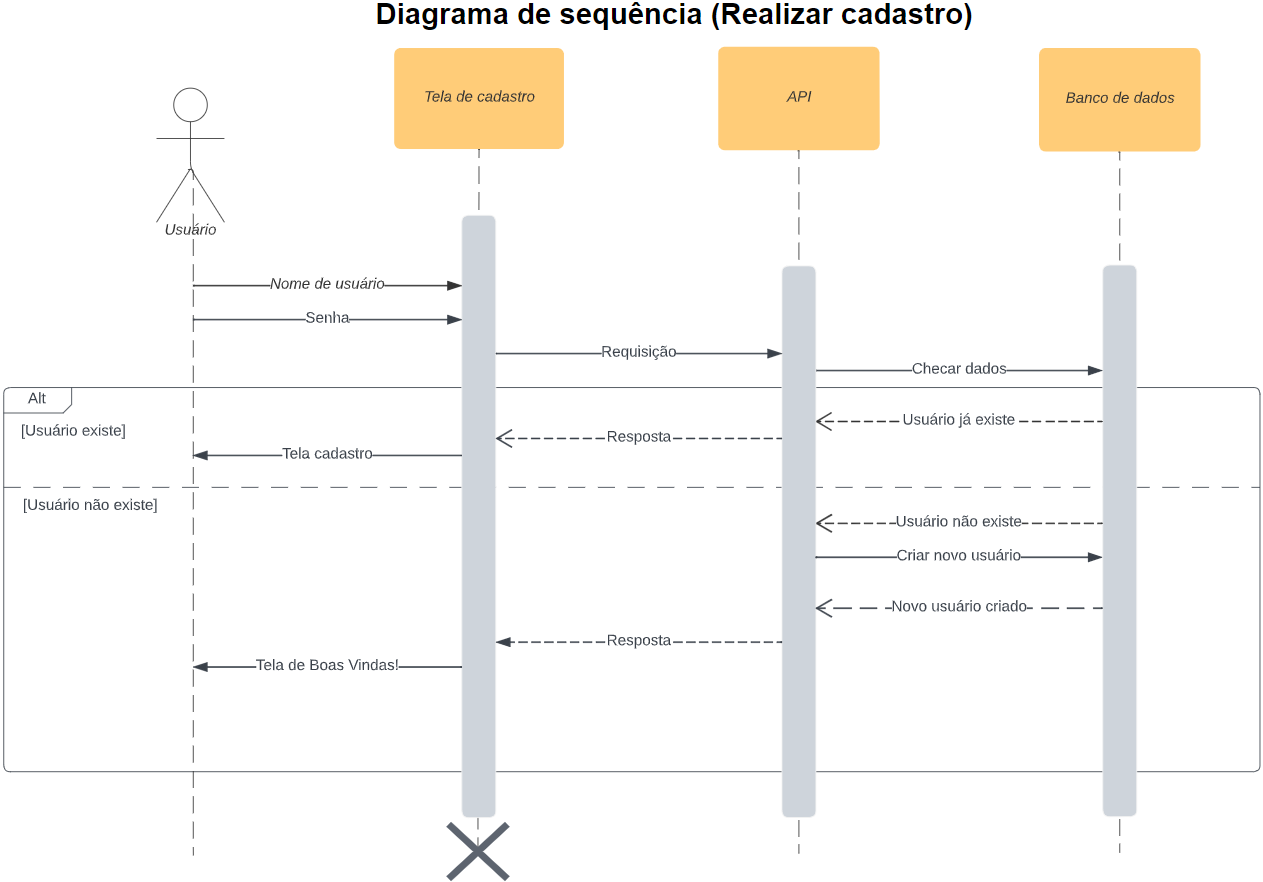


Figura 5 – Diagrama de sequência.

### Introdução a JSF (Java Server Faces)

JavaServer Faces (JSF) é a tecnologia padrão Java para construir interfaces web baseadas em componentes e orientadas a eventos. Assim como o JavaServer Pages (JSP), o JSF permite acesso a dados e lógica do lado do servidor. Ao contrário do JSP, que é essencialmente uma página HTML imbuída de recursos do lado do servidor, o JSF é um documento XML que representa componentes formais em uma árvore lógica. Os componentes JSF são suportados por objetos Java, que são independentes do HTML e possuem toda a gama de recursos Java, incluindo acesso a APIs e bancos de dados remotos. O JSF oferece facilidade de uso das seguintes formas:

Facilita a construção de uma IU usando um conjunto de componentes de IU reutilizáveis;

Simplifica a migração de dados da aplicação para a IU e provenientes dela;

Ajuda a gerenciar o estado da IU nas solicitações do servidor;

Oferece um modelo simples para conectar os eventos gerados pelo cliente ao código da aplicação do servidor;

Permite personalizar os componentes de UI para que sejam facilmente construídos e reutilizados.

A ideia chave para um framework como o JSF é encapsular (ou encapsular) tecnologias do lado do cliente como HTML, CSS e JavaScript, permitindo que os desenvolvedores construam interfaces web sem muita interação com essas tecnologias.

### Apresentação dos conceitos ORM (Object Relacional Mapping) 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. Ultimamente tem sido muito utilizada e vem crescendo bastante nos últimos anos.

Hibernate é uma ferramenta para mapeamento objeto/relacional para ambientes Java. O termo mapeamento objeto/relacional (ORM) refere-se à técnica de mapeamento de uma representação de dados em um modelo de objetos para um modelo de dados relacional baseado em um esquema E/R.

O Hibernate não cuida somente do mapeamento das classes Java para tabelas do banco de dados (e dos tipos de dados Java para os tipos de dados SQL), mas também provê facilidades para consultar e retornar os dados da consulta, e pode reduzir significativamente o tempo de desenvolvimento em contrapartida ao alto tempo gasto pelas operações manuais dos dados feitas com SQL e JDBC.

Já o JPA é um framework leve, baseado em POJOS (Plain Old Java Objects) para persistir objetos Java. A Java Persistence API, diferente do que muitos imaginam, não é apenas um framework para Mapeamento Objeto-Relacional (ORM - Object-Relational Mapping), ela também oferece diversas funcionalidades essenciais em qualquer aplicação corporativa.

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

A Estrutura de Dados pode ser definida como uma maneira de gerenciamento e organização de dados de forma que possamos realizar operações nesses dados de maneira eficiente. Estruturas de dados é fornecer ou contribuir com elementos de dados para melhor organização e armazenamento.

Existem diversos tipos de estruturas de dados muito bem catalogadas e recorrentes na ciência da computação, as quais são:

**Lista**

A ideia chave para um framework como o JSF é encapsular (ou encapsular) tecnologias do lado do cliente como HTML, CSS e JavaScript, permitindo que os desenvolvedores construam interfaces web sem muita interação com essas tecnologias.

**Fila**

Uma fila é uma estrutura de dados linear que consiste em uma coleção de itens que seguem uma sequência de primeiro a entrar, primeiro a sair. Isso implica que o primeiro item a ser inserido será o primeiro a ser removido. Você também pode dizer que os itens são removidos na ordem em que foram inseridos.

**Pilha**

Uma pilha é uma estrutura de matriz ou lista de chamadas de função e parâmetros usados ​​na programação de computadores moderna e na arquitetura da CPU. Semelhante a uma pilha de pratos em um restaurante buffet ou cafeteria, os elementos em uma pilha são adicionados ou removidos do topo da pilha, em uma ordem “último a entrar, primeiro a sair” ou LIFO.

Segue abaixo o link para uma implementação de estrutura de dados (pilha) uma em Java:

<https://raw.githubusercontent.com/queirozws/wyden-software-dev/master/item11-pilha-java/pilha.java>

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

Uma árvore AVL nada mais é que uma árvore balanceada.

Segue abaixo uma implementação em Java:

// AVL tree implementation in Java

// Create node

class Node {

int item, height;

Node left, right;

Node(int d) {

item = d;

height = 1;

}

}

// Tree class

class AVLTree {

Node root;

int height(Node N) {

if (N == null)

return 0;

return N.height;

}

int max(int a, int b) {

return (a > b) ? a : b;

}

Node rightRotate(Node y) {

Node x = y.left;

Node T2 = x.right;

x.right = y;

y.left = T2;

y.height = max(height(y.left), height(y.right)) + 1;

x.height = max(height(x.left), height(x.right)) + 1;

return x;

}

Node leftRotate(Node x) {

Node y = x.right;

Node T2 = y.left;

y.left = x;

x.right = T2;

x.height = max(height(x.left), height(x.right)) + 1;

y.height = max(height(y.left), height(y.right)) + 1;

return y;

}

// Get balance factor of a node

int getBalanceFactor(Node N) {

if (N == null)

return 0;

return height(N.left) - height(N.right);

}

// Insert a node

Node insertNode(Node node, int item) {

// Find the position and insert the node

if (node == null)

return (new Node(item));

if (item < node.item)

node.left = insertNode(node.left, item);

else if (item > node.item)

node.right = insertNode(node.right, item);

else

return node;

// Update the balance factor of each node

// And, balance the tree

node.height = 1 + max(height(node.left), height(node.right));

int balanceFactor = getBalanceFactor(node);

if (balanceFactor > 1) {

if (item < node.left.item) {

return rightRotate(node);

} else if (item > node.left.item) {

node.left = leftRotate(node.left);

return rightRotate(node);

}

}

if (balanceFactor < -1) {

if (item > node.right.item) {

return leftRotate(node);

} else if (item < node.right.item) {

node.right = rightRotate(node.right);

return leftRotate(node);

}

}

return node;

}

Node nodeWithMimumValue(Node node) {

Node current = node;

while (current.left != null)

current = current.left;

return current;

}

// Delete a node

Node deleteNode(Node root, int item) {

// Find the node to be deleted and remove it

if (root == null)

return root;

if (item < root.item)

root.left = deleteNode(root.left, item);

else if (item > root.item)

root.right = deleteNode(root.right, item);

else {

if ((root.left == null) || (root.right == null)) {

Node temp = null;

if (temp == root.left)

temp = root.right;

else

temp = root.left;

if (temp == null) {

temp = root;

root = null;

} else

root = temp;

} else {

Node temp = nodeWithMimumValue(root.right);

root.item = temp.item;

root.right = deleteNode(root.right, temp.item);

}

}

if (root == null)

return root;

// Update the balance factor of each node and balance the tree

root.height = max(height(root.left), height(root.right)) + 1;

int balanceFactor = getBalanceFactor(root);

if (balanceFactor > 1) {

if (getBalanceFactor(root.left) >= 0) {

return rightRotate(root);

} else {

root.left = leftRotate(root.left);

return rightRotate(root);

}

}

if (balanceFactor < -1) {

if (getBalanceFactor(root.right) <= 0) {

return leftRotate(root);

} else {

root.right = rightRotate(root.right);

return leftRotate(root);

}

}

return root;

}

void preOrder(Node node) {

if (node != null) {

System.out.print(node.item + " ");

preOrder(node.left);

preOrder(node.right);

}

}

// Print the tree

private void printTree(Node currPtr, String indent, boolean last) {

if (currPtr != null) {

System.out.print(indent);

if (last) {

System.out.print("R----");

indent += " ";

} else {

System.out.print("L----");

indent += "| ";

}

System.out.println(currPtr.item);

printTree(currPtr.left, indent, false);

printTree(currPtr.right, indent, true);

}

}

// Driver code

public static void main(String[] args) {

AVLTree tree = new AVLTree();

tree.root = tree.insertNode(tree.root, 33);

tree.root = tree.insertNode(tree.root, 13);

tree.root = tree.insertNode(tree.root, 53);

tree.root = tree.insertNode(tree.root, 9);

tree.root = tree.insertNode(tree.root, 21);

tree.root = tree.insertNode(tree.root, 61);

tree.root = tree.insertNode(tree.root, 8);

tree.root = tree.insertNode(tree.root, 11);

tree.printTree(tree.root, "", true);

tree.root = tree.deleteNode(tree.root, 13);

System.out.println("After Deletion: ");

tree.printTree(tree.root, "", true);

}

}

### Introdução de estrutura de dados grafos e exemplos

Um grafo é uma estrutura de dados não linear que consiste em nós e arestas. Os nós às vezes também são chamados de vértices e as arestas são linhas ou arcos que conectam quaisquer dois nós no grafo. Mais formalmente, um gráfico pode ser definido como, Um Grafo consiste em um conjunto finito de vértices (ou nós) e um conjunto de Arestas que conectam um par de nós.

Diagrama

Descrição gerada automaticamente

Os grafos são usados ​​para resolver muitos problemas da vida real. Os gráficos são usados ​​para representar redes. As redes podem incluir caminhos numa cidade ou rede telefónica ou rede de circuitos. Os grafos também são usados ​​em redes sociais como linkedIn, Facebook. Por exemplo, no Facebook, cada pessoa é representada com um vértice (ou nó). Cada nó é uma estrutura e contém informações como ID da pessoa, nome, gênero, localidade etc.

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

**Problemática do caminho com custo mínimo**

O problema do caminho 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 cacheiro viajante**

Suponha que um caixeiro viajante tenha de visitar n cidades diferentes, iniciando e encerrando sua viagem na primeira cidade. Suponha, também, que não importa a ordem com que as cidades são visitadas e que de cada uma delas pode-se ir diretamente a qualquer outra.

O problema do caixeiro viajante consiste em descobrir a rota que torna mínima a viagem total.

Exemplificando o caso n = 4:

se tivermos quatro cidades A, B, C e D, uma rota que o caixeiro deve considerar poderia ser: saia de A e daí vá para B, dessa vá para C, e daí vá para D e então volte a A. Quais são as outras possibilidades ? É muito fácil ver que existem seis rotas possíveis:

ABCDA

ABDCA

ACBDA

ACDBA

ADBCA

ADCBA

**Algoritmo Dijkstra**

O Algoritmo de Dijkstra (E.W. 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.

Algoritmo: Seja G(V,A) um grafo orientado e s um vértice de G:

1) Seja G(V,A) um grafo orientado e s um vértice de G:

2) Atribua valor zero à estimativa do custo mínimo do vértice s (a raiz da busca) e infinito às demais estimativas;

3) Atribua um valor qualquer aos precedentes (o precedente de um vértice t é o vértice que precede t no caminho de custo mínimo de s para t);

4) Enquanto houver vértice aberto:

a) seja k um vértice ainda aberto cuja estimativa seja a menor dentre todos os vértices abertos;

b) feche o vértice k;

c) Para todo vértice j ainda aberto que seja sucessor de k faça:

i) some a estimativa do vértice k com o custo do arco que une k a j;

ii) caso esta soma seja melhor que a estimativa anterior para o vértice j, substitua-a e anote k como precedente de j.

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

Em Java, nós temos 8 tipos primitivos:

* byte;
* short;
* int;
* long;
* boolean;
* char;
* float; e
* double.

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

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

Operador em java é um símbolo que é usado para executar operações. Por exemplo: +, -, \*, / etc.

Existem muitos tipos de operadores em Java que são fornecidos abaixo:

* Operador Unário,
* Operador aritmético,
* Operador shift,
* Operador relacional,
* Operador bit a bit (bitwise),
* Operador lógico,
* Operador ternário e
* Operador de atribuição.

Como estruturas de seleção nós temos:

* If / else;
* Switch / case;

Como estruturas de repetição nós temos, por exemplo:

* - For;
* - While;

Arrays

Os 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. Abaixo é mostrado se dá esse acesso aos seus elementos, lembrando que sempre sua numeração começa em 0.

Exemplo: String[] Nomes;

Funções

Em Java as funções podem ser equiparadas aos métodos das classes.

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

Foi desenvolvido um backend para utilizando o framework Spring Boot.

Segue abaixo o link para o código hospedado no Github bem como a explicação do backend em vídeo:

Código: <https://github.com/queirozws/wyden-software-dev/tree/master/item16-backend-java>

Explicação em vídeo: <https://drive.google.com/file/d/18d6PdtI9pLjOMRGX-e5a38yuEEE0kKUF>

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

As questões escolhidas para resolução são da lista da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS).

i. Exercício 1 (questão 8) – pares;

ii. Exercício 2 (questão 30) – primos;

iii. Exercício 3 (questão 51) – fatorial.

A resoluções encontram-se no endereço eletrônico a seguir: <https://github.com/queirozws/wyden-software-dev/tree/master/item16-backend-java>

REFERÊNCIAS

Visão geral do cliente-servidor. MDN Web Docs, 2022. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/First_steps/Client-Server_overview>>. Acesso em 05 jun. 2022.

Qual a diferença entre página web, site, servidor web e mecanismo de busca?. MDN Web Docs, 2022. Disponível em: <<https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Pages_sites_servers_and_search_engines>>. Acesso em: 03 jun. 2022.

Navegador. MDN Web Docs, 2022. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/Glossary/Browser>>. Acesso em: 03 jun. 2022.

Servidor. MDN Web Docs, 2022. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/Glossary/Server>>. Acesso em: 03 jun. 2022.

O que é o JSP? Uma introdução ao JavaServer Pages. InfoWorld, 2022. Disponível em: <<https://www.infoworld.com/article/3336161/what-is-jsp-introduction-to-javaserver-pages.html>>. Acesso em: 03 jun. 2022.

Java Server Pages. Dev Media, 2022. Disponível em: < <https://www.devmedia.com.br/java-server-pages/7245>>. Acesso em: 03 jun. 2022.

Introdução ao JPA – Java Persistente API. Dev Midia, 2022. Disponível em: <<https://www.devmedia.com.br/introducao-a-jpa-java-persistence-api/28173>>. Acesso em: 03 jun. 2022.

ORM: Object Relational Mapper. Dev Media, 2022. Disponível em: <<https://www.devmedia.com.br/orm-object-relational-mapper/19056>>. Acesso em: 03 jun. 2022.

XXXXX. MDN Web Docs, 2022. Disponível em: <<http://www.dainf.ct.utfpr.edu.br/petcoce/wp-content/uploads/2011/06/DiscreteMathFloydWarshall.pdf>>. Acesso em: 03 jun. 2022.

Problema do caixeiro viajante. UFGRS, 2022. Disponível em: <<http://www.mat.ufrgs.br/~portosil/caixeiro.html>>. Acesso em: 03 jun. 2022.

Java. Dev Media, 2022. Disponível em: <<https://www.devmedia.com.br/java-variaveis-e-constantes/38311>>. Acesso em: 03 jun. 2022.

Funções em Java. GUJ, 2022. Disponível em: <<https://www.guj.com.br/t/funcoes-em-java/78220>>. Acesso em: 03 jun. 2022.

Disponível em: <https://pt.gadget-info.com/difference-between-cgi>. Acesso em: 05 jun. 2022.

Disponível em: <<https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1>>. Acesso em: 04 jun. 2022.

Disponível em: <<https://www.geeksforgeeks.org/queue-data-structure/>>. Acesso em: 04 jun. 2022.