Centro Universitário Unifanor

Murilo Carvalho de Freitas (202051901721)

Lucas Duarte Colares (202051475511)

Desenvolvimento de Software

Atividade Avaliativa

Fortaleza

2022

1. Arquitetura Web: Fundamentos; Modelos Arquiteturais

A arquitetura de software de um sistema consiste na definição dos componentes de software, buscando suas propriedades externas e seus relacionamentos com outros software visando uma solução fácil para um problema.

Existe uma divisão em camadas para auxiliar nesta tomada de decisão da arquitetura para tornar os sistemas mais flexíveis e reutilizáveis. Cada camada ou componentes vai definir uma funcionalidade do sistema.

Os modelos arquiteturais se consistem: No modelo monolítico sendo o modelo mais comum e usada no desenvolvimento web e o mais antigo, em que o sistema, o banco de dados e o servidor local, são executados em uma única máquina; O modelo cliente-servidor, que se consiste em uma aplicação distribuída, ou seja, na rede existem os fornecedores de serviços a rede, que são chamados de servidores, e existe quem requer dos serviços do servidor, no caso o cliente. O cliente não compartilha seus recursos com o servidor, mas ele solicita alguma função do servidor, sendo assim responsável por iniciar a comunicação com o servidor: Temos o modelo Peer to peer, também conhecido como P2P, que consiste em uma arquitetura de redes em que cada par coopera entre si para prover serviços um ao outro, um sistema para compartilhamento de arquivos, documentos e informações, sem a necessidade de um servidor central.

2. HTML: Exemplo de construção de cadastro de formulário

https://drive.google.com/file/d/1pAarPez74h6lsVjPZkgLPykCN79ujgm2/view?usp=sharing

https://github.com/MuriloCarvs/atvd-desenvolvimento-de-software

3. Descrever as diferenças entre Servlets e Scripts CGI

Os Servlets são implementações em Java compilados no bytecode de Java e executado na JVM, já os Scripts CGI são gravados no sistema operacional e armazenados em determinado diretório.

Outra diferença é que o CGI é específico da plataforma, o que acaba dificultando a alternância entre os sistemas operacionais, já os Servlets são independente da plataforma, basta ter instado o JVM em qualquer sistema operacional para ser executado.

O Servlets é portátil, enquanto o CGI não..

As solicitações do cliente do Servlet não são separados em processos desnecessários, eles compartilham espaço na memória da JVM, já o CGI pode gerar processo separados.

Os Servlets são compilados para o bytecode Java que é executado na JVM, já o CGI pode ser executado no sistema operacional nativo do servidor.

Os Servlets traduzem e compilam o programa e o processa, já o CGI podem ser processados de forma direta.

O Servlets é mais seguro por ser em Java.

O Servlets é mais rápido, além do seu desempenho e eficiência também serem superiores a CGI.

4. Introdução ao JSP (Java Server Pages)

A JSP é uma linguagem gratuita criada pela SUN, uma linguagem de script com especificação aberta com o objetivo de criar páginas Web dinâmicas, ao invés de usar o HTML para criar páginas estáticas, porém também é possível escrever HTML com códigos JSP embutidos, assim dando uma responsabilidade ao JSP deixar dinâmica as páginas estáticas do HTML.

O JSP necessita de um servidor para funcionar por ser uma linguagem Serve-side script, o usuário não consegue ver a codificação JSP, pois ela é convertida diretamente pelo servidor, sendo apresentado ao usuário apenas a codificação HTML.

Por ser uma linguagem gratuita e possuir especificação aberta possui diversos servidores que suportam ela, como o Tomcat, GlassFish, JBoss e entre outros.

5. Introdução aos Padrões de Projetos; explicar funcionamento arquitetura MVC

Os padrões de projetos consistem em soluções já testadas para problemas recorrentes no desenvolvimento de software, que deixam seu código mais manutenível, pois essas soluções se baseiam em um baixo acoplamento, eles são divididos em categorias como:
Padrões de Projetos de Criação, onde eles resolvem os problemas relacionados a escrita de classes tanto na criação de classes como na criação de objetos através da Delegation;
Padrões de Projetos Estrutura, onde ele explica como montar classes e objetos em estruturas maiores mas ainda mantendo essas estruturas flexíveis e eficientes;
Padrões de Projetos de Comportamento, onde são voltados aos algoritmos e a designação de responsabilidade entre objetos.

A arquitetura MVC é um padrão de arquitetura de software dividido em três camadas, a Model, View e Controller, onde o usuário acessa uma aplicação web em que o navegador (client) faz uma requisição HTTP que vai chegar ao Controller em seguida após ele identificar a requisição comunica ao Model, onde ele vai fazer uma requisição ao banco de dados e o banco de dados retorna os dados de volta para o Model, em seguida o Model retorna ao Controller, o Controller segue para a View, onde ela vai renderizar os dados, com os dados rederidados eles retornam ao Controller e o Controller faz outra requisição HTTP para o navegador com as informações completas.

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

A Programação Orientada a Objetos, também conhecido como POO, tem como base um conceito de objeto, então um sistema orientado a objeto é um conjunto de objetos que representam os conceitos do mundo real, interagindo computacionalmente com as mesmas características e comportamentos reais.

As classes são uma forma de definir um tipo de dado em uma linguagem orientada a objeto, sendo formada por dados e comportamentos, assim para a definição deste tipo de dado são utilizados os atributos e para definir o comportamento são utilizados os métodos. Os atributos são as informações e características do objeto, como por exemplo em um carro, a cor, o modelo, a qualidade de rodas e etc. E os métodos, são o comportamento do carro, como a função de acelerar, frear, abrir portas e etc.

Os objetos representam uma cópia criada a partir de uma classe e pertence sempre a uma única classe, mas uma classe pode ter vários objetos criados.

Os atributos ou propriedades são aquilo que descreve um intervalo de valores que as instâncias de uma classe podem apresentar.

Os métodos são funções que realizam as ações próprias do objeto, sendo as ações que o objeto pode realizar.

Os níveis de visibilidade são distribuídos em três tipos, a private que é uma função local e um único bloco de código, a public que é visível para tudo, uma função que pode ser chamada em qualquer momento. E a protect, neste caso é restringido um parâmetro fora da classe, mas ainda acessível as suas subclasses.

A herança é um tipo de relacionamento entre classes, que uma classe é herda características de outra classe.

O encapsulamento se trata do fato de esconder as propriedades, criando uma espécie de caixa preta.

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 extends é uma palavra reservado para criar herança entre classes.

7. Introdução a linguagem UML.

A UML (Unified Modeling Langue) é uma linguagem unificada que habilita profissionais de TI a modelar e documentar aplicações de software, ela oferece um meio de visualizar a arquitetura de um sistema por meio de diagramas, incluindo atividades, componentes individuais do sistema, a interação desses componentes, interfaces, interação com o mundo externo e entre outras. Assim os diagramas podem ser representados por duas visões distintas, a estática ou estrutural, que apresenta a estrutura estática por meio de objetos, operações, relações e atributos. Já a dinâmica ou comportamental apresentando um comportamento dinâmico por meio de colaboração entre os objetos e mudanças de seus estados internos, tendo como mais importantes os diagramas de caso de uso, diagramas de classes, diagramas de sequência, diagramas máquina de estados e diagramas de atividades.

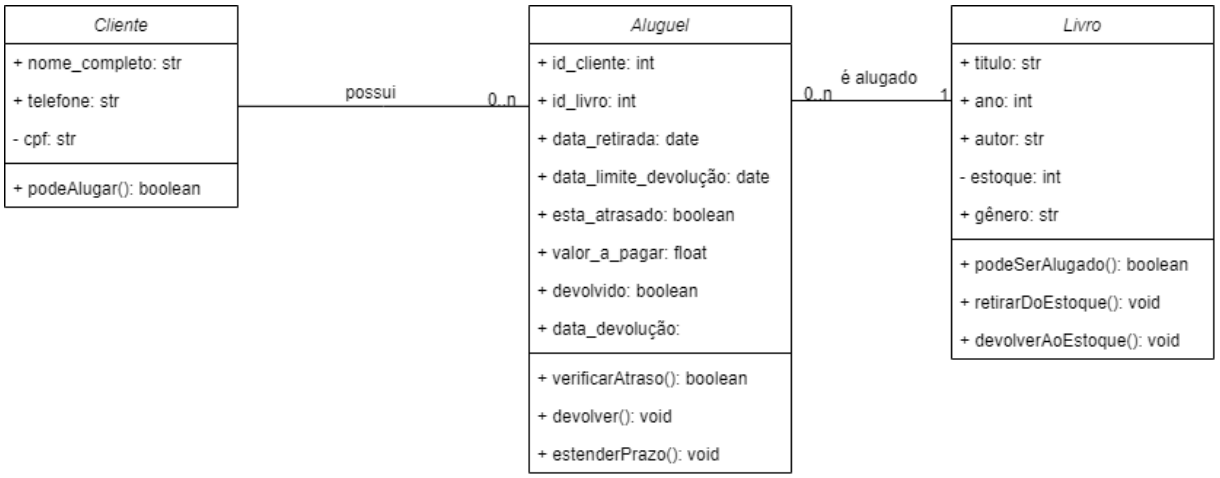
8. Modelar um domínio (livre escolha), apresentar caso de uso, diagrama de classe e sequencial

Escolhemos modelar um domínio de uma biblioteca, onde o caso uso da aplicação seria armazenar e gerenciar dados dos clientes da biblioteca e de seus livros, assim como registrar aluguéis de livros, se estes já foram devolvidos, se estão atrasados e qual valor deve ser pago na devolução. O sistema seria usado pelos bibliotecários, que teriam acesso a esses dados.

Um livro só pode ser alugado se a biblioteca tiver ao menos uma cópia em estoque.

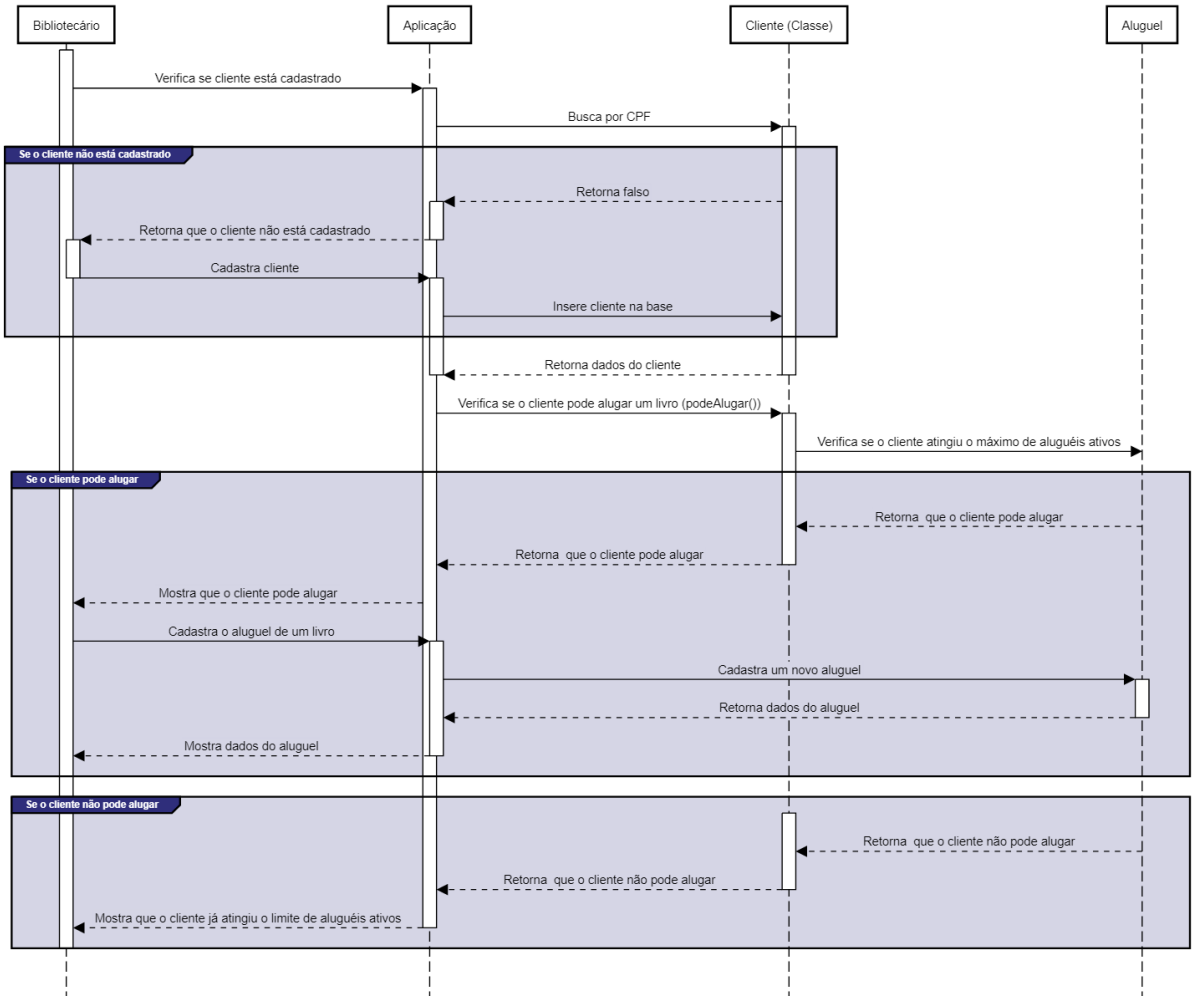
Um cliente só pode ter até 3 aluguéis ativos no momento, mas não tem limites de aluguéis totais ao longo do tempo.

Abaixo temos o diagrama de classe da aplicação, que foi feito usando a ferramenta draw.io:

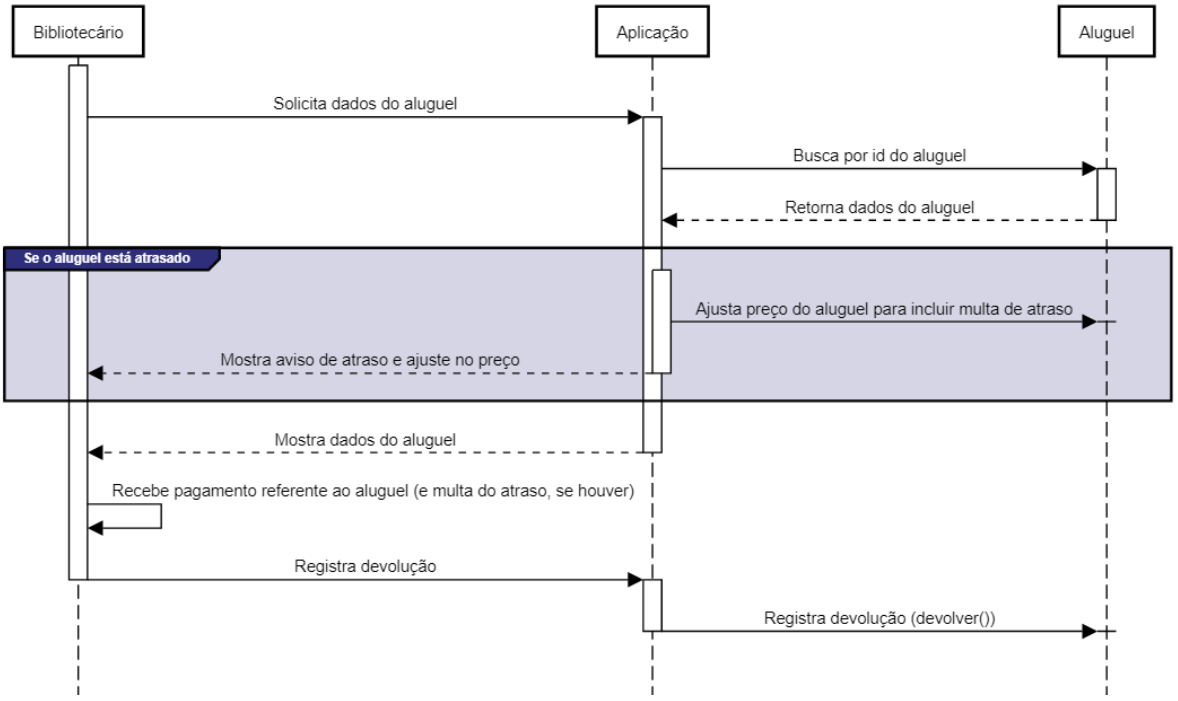


E abaixo temos os diagramas de sequência referentes ao fluxo de aluguel e devolução de um livro, feitos usando a ferramenta https://sequencediagram.org:

Aluguel de um livro



Devolução de um livro



9. Introdução a JSF (Java Server Faces)

O primeiro contato da linguagem de programação Java com o desenvolvimento de aplicações web foi por meio dos Servlets, que já abordamos anteriormente. O uso dos Servlets serviu muito bem para o período inicial, mas com a constante expansão da internet e o crescimento do número de acessos e consequentemente das demandas fez com que alguns problemas nesta arquitetura fossem expostos.

Como um dos principais problemas enfrentados pelo uso de Servlets podemos citar a dificuldade de gerar e devolver ao cliente páginas em HTML, visto que a arquitetura inicialmente respondia apenas texto puro e gerar e devolver uma página HTML dinâmica (que se adeque aos dados e condições da requisição e resposta) requer uma grande complexidade para implementar em arquiteturas assim. Além disso, o fato de que em aplicações Servlets muitas lógicas e responsabilidades ficavam concentradas em um único artefato tornava difícil a implementação e manutenção de sistemas complexos.

Surgiu então a arquitetura Java Server Faces, que já havia sido idealizada e desenvolvida com o ideal de facilitar a implementação de interfaces para o usuário, e evitar os mesmos erros dos Servlets.

Nessa tecnologia, os desenvolvedores tinham acesso a uma variada gama de componentes para páginas web, além disso, o JSF trazia também uma melhor separação das camadas da aplicação, facilitando o desenvolvimento de sistemas que se tornavam cada vez mais complexos.

Usando o JSF, programadores podiam escrever páginas em XHTML e conecta-las à classes Java por meio do uso de Data Binding, assim algumas funcionalidades tornavam-se mais fáceis de implementar, como por exemplo, exibir dados de um banco de dados em uma tabela HTML.

O JSF também trouxe ao longo de seu ciclo de vida outras funcionalidades, como a possibilidade de implementar soluções Ajax sem que os programadores precisassem usar Javascript e a funcionalidade Facelets, um tipo de template próprio para definição de estruturas padrões para as páginas de uma aplicação.

O JSF tornou-se tão usado que surgiram frameworks para essa tecnologia, como o PrimeFaces, que trazia mais de 100 novos componentes com visuais modernos e configuráveis sem a necessidade do desenvolvedor saber usar CSS ou Javascript.

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

O uso de banco de dados tornou-se cada vez mais comum ao longo da história do desenvolvimento de sistemas, afinal, a habilidade de rapidamente armazenar, consultar, atualizar e apagar dados em uma memória facilita o trabalho de diversos funcionários em variadas funções nos mais diferentes ramos da nossa sociedade. Um sistema usado em uma clínica médica precisa ser capaz de guardar as consultas agendadas, remarcar a consulta ou cancelá-la, por exemplo.

Os desenvolvedores desses sistemas precisam integrar a camada de negócios ao banco de dados da aplicação, para que assim, a aplicação consiga realizar operações no banco conforme o usuário utiliza a aplicação. Por exemplo: quando o usuário do sistema de clínicas citado anteriormente marca uma consulta, precisamos criar o registro desta consulta no banco de dados.

Porém, a integração não é simples, principalmente por causa da diferença de paradigmas, afinal, os bancos de dados mais comuns são os relacionais, que apresentam uma estrutura baseada em tabelas e colunas, relacionamentos por meio de chaves e tabelas pivô, etc… e as linguagens de programação em sua maioria são orientadas a objeto, apresentando uma estrutura de classes, objetos, atributos, métodos, etc…

Uma solução que foi usada por bastante tempo era a de escrever as *queries* (instruções para o banco de dados) para funções específicas (Ex.: cadastrar uma clínica) e executá-las de acordo com a necessidade. Essa estratégia atende muito bem sistemas simples, mas a partir do momento em que a lógica de negócio torna-se complexa, também tornam-se o banco de dados e consequentemente as instruções. Realizar manutenções em *queries* longas pode levar muito tempo e trabalho.

Para solucionar esses e outros problemas, foram criados os ORMs, bibliotecas ou frameworks que facilitam a integração de uma linguagem de programação orientada a objetos à um SGBD (Sistema Gerenciador de Banco de Dados). Os ORMs definem uma estrutura a ser usada na linguagem de programação e a “traduz” para *queries* assim como também as executa e devolve seus resultados.

O uso de ORMs diminui a complexidade do projeto, assim como reduz o tempo de desenvolvimento além de aprimorar a legibilidade do código e tornar mais fácil para que novos desenvolvedores entendam as regras de negócios implementadas no banco de dados.

O *Java Persistence Api* (JPA) é um framework Java que como o nome indica, é voltado para a persistência de dados, ele é hoje um dos mais famosos frameworks por apresentar uma estrutura simples, baseadas em POJOS (*Plain Old Java Objects*), ou seja, uma estrutura simples sem a necessidade de pensar em relacionamentos como em um banco de dados.

O JPA surgiu para unificar e regrar os diferentes ORMs que foram criados para a linguagem Java ao longo do tempo, antes de sua criação cada ORM seguia uma estrutura própria, o que era um problema para ambos os desenvolvedores e a linguagem Java em si. O JPA criou regras e normas para a estrutura desses, facilitando o aprendizado e manutenção de códigos que os usavam.

O *Hibernate* é também um conhecido framework para persistência de dados, pois foi um dos primeiros a implementar um ORM em sua estrutura além de ser o líder na criação do JPA, pois grande parte das regras do JPA foram inspiradas na estrutura do Hibernate, que era o melhor framework na época de acordo com os criadores do JPA.

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

Estrutura de dados é o ramo da computação que estuda os diversos mecanismos de organização de dados para atender a diversas necessidades computacionais. Ao longo do tempo, foram criadas algumas estruturas de dados que são usadas até hoje, sendo as principais: listas, filas e pilhas.

Os dois tipos de listas mais conhecidos são as listas ligadas e as duplamente ligadas, mas para explicar melhor cada uma, preciso explicar o que é uma lista: um conjunto de nós, sendo que cada nó guarda a informação necessária para continuar a lista. Em uma lista ligada, a lista em si guarda apenas o endereço de seu primeiro nó e cada nó guarda o endereço do próximo nó da lista (se houver), assim, para identificar o último nó basta apenas continuarmos indo de nó em nó até acharmos um que não aponte para nenhum próximo. Em uma lista duplamente ligada, os nós guardam além de seu sucessor, o endereço de seu antecessor, o que facilita a navegação pela lista, e de forma semelhante, o último nó não possui sucessor assim como o primeiro não possui antecessor.

Pilhas e filas podem ser classificadas como listas, mas com funções de adicionar e remover itens ajustadas para sua implementação. Em uma pilha temos o comportamento conhecido como FILO (First in, Last Out) ou seja, os itens são adicionados no topo da pilha assim como também são removidos do topo, de forma que o primeiro item a entrar (first item), fica na parte mais baixa da pilha e é o último a ser removido (Last Out).

Já as filas seguem o comportamento FIFO (First In, first out), ou seja, os itens são removidos de acordo com a ordem de chegada, em outras palavras, o primeiro item a entrar na fila (First In) é o primeiro a ser removido (First Out).

Abaixo temos o exemplo de implementação de uma fila em Java, onde temos uma classe com os atributos e métodos necessários para simular o comportamento:

Escolhemos implementar a estrutura de dados Fila em Java, segue link para a pasta do repositório onde temos tanto a implementação em si no arquivo “Fila.java” quanto um script para testa-la no arquivo “Main.java”:
https://github.com/MuriloCarvs/atvd-desenvolvimento-de-software/tree/main/q11

Abaixo também temos o link para um vídeo explicando melhor sobre o tópico:
https://drive.google.com/file/d/1_V7AREc7K-BrZFgq6-_LZvnf5Nf-ByEk/view?usp=sharing

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

É chamada de árvore AVL uma árvore binária de pesquisa que possui a capacidade de se auto-ordenar, em outra palavras, quando itens são adicionados ou removidos é executado um algoritmo que verifica se as sub-árvores esquerda e direita possuem alturas iguais ou próximas, e se necessário e possível, rearranja os itens de forma que se alcance esse resultado.

Abaixo temos um exemplo de implementação de uma árvore AVL em Java. Note que o método *insert*, responsável pela inserção de um item na árvore realiza a verificação do balanceamento da árvore e se identificar a necessidade, reordena os itens de forma a alcançar um balanço.

Segue link da implementação de uma árvore AVL juntamente de um script para comprovar seu funcionamento:
https://github.com/MuriloCarvs/atvd-desenvolvimento-de-software/tree/main/q12

13. Introdução de estrutura de dados grafos e exemplos

É definido como grafo uma estrutura formada por vértices e arestas que ligam estes, essa estrutura é bastante comum em nosso dia a dia e podemos citar como exemplos: rotas de vôo e escalas, mapas de estradas, rotas de metrôs, redes sociais, etc…

Os grafos podem ser divididos em orientados e não-orientados, no primeiro tipo as arestas seguem um sentido único, ou seja, o deslocamento por essa aresta só pode seguir de um dos vértices para o outro, nunca ao contrário. Por exemplo: em um grafo onde temos os vértices A, B, C e D, temos as arestas A->B, B->C, C->D e D->A, nesse grafo, se começarmos no vértice A, só poderemos ir para o B, e do B para o C, e assim por diante, mas nunca poderemos fazer algo como ir do B para o A. O segundo tipo não possui essa restrição e o deslocamento pode ser feito em ambos os sentidos.

Temos ainda os grafos valorados, que possuem valores para as arestas e/ou para os vértices, por exemplo: se criarmos um grafo de estradas entre cidades, o valor das arestas seriam a distância entre as 2 cidades (vértices), e se criarmos um grafo de uma rede social, o valor do vértice poderia ser o nome de uma pessoa.

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

Problemática do caminho com custo mínimo é o nome dado a situações onde precisamos encontrar o menor caminho ou o de menor custo entre dois pontos, sendo que temos vários possíveis caminhos e rotas. Esses problemas geralmente são encontrados em estruturas de dados como grafos, onde temos vários pontos (vértices) e caminhos (arestas) que os ligam e onde um ponto pode se conectar a vários outros por varios caminhos.

O problema do caixeiro viajante é algo semelhante, é o nome dado a questões da seguinte forma: “Dado uma lista de cidades e a distância entre cada par de cidades, qual a menor rota possível que um caixeiro poderia fazer para passar por todas as cidades e retornar a sua cidade de origem?”. Mais uma vez, esses problemas geralmente são encontrados em grafos, onde os vértices são as cidades e as arestas (valoradas) representam as ligações e as distâncias entre determinado par de cidades.

Criado pelo Dr. Edsger W. Dijkstra, o algoritmo Dijkstra serve para encontramos o menor caminho possível partindo de um nó de um grafo e passando por todos os outros nós. O algoritmo funciona da seguinte forma: dado um grafo com N elementos e um destes elemento para começarmos, inicialmente “fechamos” este elemento, ou seja, registramos que passamos por ele e que não precisaremos voltar a ele no futuro, e avaliamos a distância entre este elemento e todos os outros que ele está conectado, escolhemos o de menor custo, nos movemos para ele e o fechamos, então, repete-se esse processo até que tenha-se passado por todos os elementos do grafo, ou seja, todos estão “fechados”.

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

Os tipos primitivos são os tipos mais básicos de dados que podemos ter em uma linguagem, no caso da linguagem Java temos: *int* (números inteiros que cabem em 32 bits)*, byte* (números inteiros que cabem em 8 bits)*, short* (números inteiros que cabem em 16 bits)*, long* (números inteiros que cabem em 64 bits)*, float* (números de ponto flutuante, que cabem em até 32 bits)*, double* (números de ponto flutuante com dupla precisão que cabem em até 64 bits)*, boolean* (veradeiro ou falso) *and char* (um único caractere)*.*

Uma variável é um nome simbólico dado a um espaço reservado da memória que pode ser manipulado durante a execução do programa, em linguagens fortemente tipadas como o Java uma variável recebe um tipo de dado em sua criação e só aceitará dados daquele tipo para serem guardados.

Uma constante é semelhante a uma variável, com a única diferença sendo que uma vez inicializada com um valor, uma constante não pode ser mais alterada, apenas lida.

Estruturas de seleção são instruções de uma linguagem de programação que servem para executar uma determinada parte do código somente se uma ou mais condições forem cumpridas. Em Java temos o *if* (executa o código se as condições colocadas forem cumpridas), o *else* (executa o seu bloco de código se as condições de um *if* antecessor falharem), a junção dessas duas, o *else if* (se o *if* antecessor falhar, executa o código somente se a sua própria condição for cumprida) e o *switch* com os *cases* (estrutura que compara as condições dos *cases* com a expressão colocada no *switch*, executando os que as condições se cumprirem, tendo a possibilidade de colocarmos um código padrão a ser executado em todos os casos, mesmo que nenhum *case* se cumpra).

As estruturas de repetição, como o nome indica, servem para repetir a execução de determinado bloco de código. Em java temos o *while* (executa o código enquanto a expressão recebida for verdadeira), o *do while* (obrigatoriamente executa o código uma vez e o repete somente se a expressão recebida for verdadeira) e o *for* (se apresenta de duas maneiras, em uma, podemos percorrer uma lista de valores e ir atribuindo cada um dos itens a uma variável especificada e em outra, podemos fazer uma estrutura dividida em três partes, onde declaramos ou alteramos uma variável, definimos uma condição e por último definimos uma expressão simples a ser executada a cada repetição, essa estrutura é geralmente usada para repetir o código um determinado número de vezes).

Array em Java é um tipo de dado complexo, que pode guardar até N valores do mesmo tipo. O tipo de dado e o número de elementos de um *array* são definidos em sua inicialização, pode-se também já definir os valores dos elementos. Um *array* guarda os valores em índices numerados que começam do 0, e por meio desses índices é possível acessar e manipular os valores, mas não mexer na estrutura do array, seu número de elementos ou seu tipo de dado.

Functions, ou funções, são blocos de códigos executáveis que podem receber parâmetros e retornar valores. Na declaração de uma *function* definimos o tipo de dado de seu retorno (usando *void* caso ela não for retornar nada) e de seus parâmetros (se houver algum), além da sua visibilidade dentro da classe. Funções servem para evitarmos a repetição de código, criando uma função que executa uma tarefa que vá ser repetida várias vezes durante a execução e servem também dentro da Programação Orientada a Objetos para representarmos comportamentos de uma determinada classe.

16. Apresentar um exemplo de backend em JAVA utilizando STS Spring Boot

Criamos uma aplicação básica em Spring Boot, com um único endpoint que recebe dois números (*n1* e *n2*) no corpo de uma requisição HTTP POST e retorna a soma dos dois números.

O código da aplicação está disponível em: https://github.com/MuriloCarvs/atvd-desenvolvimento-de-software/tree/main/q16

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

Lista escolhida: UFBA.

As implementações estão disponível no repositório do GitHub: https://github.com/MuriloCarvs/atvd-desenvolvimento-de-software.

Abaixo seguem o link para a implementação de cada exercício escolhido:

- Questão 2: https://github.com/MuriloCarvs/atvd-desenvolvimento-de-software/tree/main/q17/Q2
- Questão 3: https://github.com/MuriloCarvs/atvd-desenvolvimento-de-software/tree/main/q17/Q3
- Questão 7: https://github.com/MuriloCarvs/atvd-desenvolvimento-de-software/tree/main/q17/Q7