Classe Main

Essa aqui é onde tudo começa. O main() é tipo o portão de entrada do programa.

Scanner scan = new Scanner(System.in);

Serve pra ler tudo que o usuário digitar. Como os comandos vêm um por linha, a gente usa nextLine().

Btree a = new Btree();

Aqui eu criei a árvore binária de busca. Ela vai guardar os alunos que forem sendo inseridos.

int qo = Integer.parseInt(scan.nextLine());

Lê quantos comandos o usuário vai digitar. Isso controla o loop de leitura depois.

for (int i = 0; i < qo; i++)

Roda o loop a quantidade de vezes que foi informada. Cada vez lê um comando e trata ele.

INSERIR

Pega o ID, nome e as notas. Dá um split() na linha e transforma as notas em double[]. Depois cria um objeto Aluno e manda inserir na árvore com a.add(aluno);.

BUSCAR

Lê o ID e chama a.show_id(id) pra procurar o aluno na árvore e mostrar ele.

LISTAR_POR_ID

Chama a.show() pra listar todos os alunos na ordem do ID.

ATUALIZAR_NOTAS

Pega o ID e as novas notas. Chama a.atualizar(id, novasNotas) pra trocar as notas do aluno com esse ID.

Classe Aluno

Só guarda os dados do aluno. Cada objeto desse tem:

id: número único que identifica o aluno.

nome: o nome do aluno.

nota[]: array com as notas dele.

Tem também os getters e setters. É só pra pegar e mudar os valores de forma mais organizada (encapsulamento mesmo).

Tipo: getId() pega o ID, getNota() pega o array de notas, etc.

Classe Btree

Essa aqui é a árvore binária de busca.

private Bnode raiz;

Começa com a raiz nula, ou seja, vazia.

add(Aluno x)

Se a árvore tiver vazia, cria a raiz. Se já tiver coisa, manda pro método add() lá no Bnode, que vai colocar o aluno no lugar certo.

show()

Mostra todos os alunos em ordem de ID, chamando show() da raiz.

show_id(int id)

Procura o aluno com aquele ID chamando show_id() da raiz.

atualizar(int id, double[] nota_nova)

Tenta achar o aluno com esse ID e, se encontrar, atualiza as notas dele.

Classe Bnode

Aqui tá o coração da árvore.

private Aluno aln;

private Bnode esq, dir;

Cada nó da árvore guarda um aluno e tem dois caminhos: um pra esquerda (ID menor) e um pra direita (ID maior).

add(Aluno x)

Compara o ID do aluno novo com o do nó atual.

Se for menor, tenta jogar pra esquerda.

Se for maior ou igual, vai pra direita.

Faz isso até achar um lugar vazio e criar o novo nó.

show()

Mostra os alunos em ordem crescente de ID:

Vai pra esquerda primeiro

Mostra o nó atual

Vai pra direita depois

show_id(int id)

Procura um aluno com o ID passado. Se achar, mostra. Se não, continua procurando.

atualizar(int id, double[] nota_nova)

Procura o aluno. Quando acha, recria o Aluno com o mesmo ID e nome, mas agora com as notas novas.