Nama : Athallah Khansa Ziven Jaya Bahtiar

Kelas : D3 RPLA 47-02

NIM : 607062300069

Class BTNode2

class BTNode2<E> {

private BTNode2<E> left;

private BTNode2<E> right;

private E data;

/\* Default Constructor \*/

public BTNode2() {

this.left = null;

this.right = null;

this.data = null;

}

/\* Constructor with data \*/

public BTNode2(E item) {

this.left = null;

this.right = null;

this.data = item;

}

/\* Setters for left and right nodes \*/

public void setLeft(BTNode2<E> left) {

this.left = left;

}

public void setRight(BTNode2<E> right) {

this.right = right;

}

/\* Getters for left and right nodes \*/

public BTNode2<E> getLeft() {

return left;

}

public BTNode2<E> getRight() {

return right;

}

/\* Setters and getters for node data \*/

public void setData(E data) {

this.data = data;

}

public E getData() {

return data;

}

}

Penjelasan :

Kodingan di atas adalah definisi dari kelas BTNode2, yang merupakan simpul dalam pohon biner. Berikut adalah penjelasan dari setiap bagian kodingan tersebut:

1. Variabel Instance:

private BTNode2<E> left; dan private BTNode2<E> right;: Menyimpan referensi ke simpul kiri dan kanan dari simpul saat ini.

private E data;: Menyimpan data yang terkait dengan simpul saat ini.

1. Constructor:

public BTNode2(): Konstruktor default yang inisialisasi simpul dengan kedua referensi ke simpul kiri dan kanan menjadi null, dan data menjadi null.

public BTNode2(E item): Konstruktor yang menginisialisasi simpul dengan data yang disediakan oleh pengguna dan kedua referensi ke simpul kiri dan kanan menjadi null.

1. Setters dan Getters:

public void setLeft(BTNode2<E> left) dan public void setRight(BTNode2<E> right): Metode untuk mengatur simpul kiri dan kanan.

public BTNode2<E> getLeft() dan public BTNode2<E> getRight(): Metode untuk mendapatkan referensi ke simpul kiri dan kanan.

public void setData(E data) dan public E getData(): Metode untuk mengatur dan mendapatkan data yang terkait dengan simpul saat ini.

Kelas BTNode2 ini menyediakan struktur dasar yang diperlukan untuk membuat dan mengelola simpul-simpul dalam pohon biner. Dengan menggunakan kelas ini, Anda dapat membuat pohon biner dengan mudah dan mengakses serta memanipulasi simpul-simpulnya.

Class BinaryTree2

class BinaryTree2<E> {

private BTNode2<E> root;

/\* Constructor \*/

public BinaryTree2() {

root = null;

}

/\* Function to check if the tree is empty \*/

public boolean isEmpty() {

return root == null;

}

/\* Function to insert data \*/

public void insert(E data) {

root = insert(root, data);

}

/\* Recursive function to insert data \*/

private BTNode2<E> insert(BTNode2<E> node, E data) {

if (node == null) {

return new BTNode2<>(data);

} else {

if (node.getLeft() == null) {

node.setLeft(insert(node.getLeft(), data));

} else {

node.setRight(insert(node.getRight(), data));

}

return node;

}

}

/\* Function to count the number of nodes \*/

public int countNodes() {

return countNodes(root);

}

/\* Recursive function to count the number of nodes \*/

private int countNodes(BTNode2<E> node) {

if (node == null) {

return 0;

} else {

return 1 + countNodes(node.getLeft()) + countNodes(node.getRight());

}

}

/\* Function to search for an element \*/

public boolean search(E val) {

return search(root, val);

}

/\* Recursive function to search for an element \*/

private boolean search(BTNode2<E> node, E val) {

if (node == null) {

return false;

}

if (node.getData().equals(val)) {

return true;

}

return search(node.getLeft(), val) || search(node.getRight(), val);

}

/\* Function for inorder traversal \*/

public void inorder() {

inorder(root);

}

private void inorder(BTNode2<E> node) {

if (node != null) {

inorder(node.getLeft());

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

inorder(node.getRight());

}

}

/\* Function for preorder traversal \*/

public void preorder() {

preorder(root);

}

private void preorder(BTNode2<E> node) {

if (node != null) {

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

preorder(node.getLeft());

preorder(node.getRight());

}

}

/\* Function for postorder traversal \*/

public void postorder() {

postorder(root);

}

private void postorder(BTNode2<E> node) {

if (node != null) {

postorder(node.getLeft());

postorder(node.getRight());

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

}

}

}

Penjelasan :

Kode di atas adalah implementasi dari struktur data pohon biner dalam bahasa pemrograman Java menggunakan generic. Ini terdiri dari kelas BinaryTree2 yang merupakan representasi dari pohon biner.

Berikut adalah penjelasan dari setiap bagian kode:

1. Variabel Instance:

private BTNode2<E> root;: Variabel yang menyimpan referensi ke akar pohon biner.

1. Constructor:

public BinaryTree2(): Konstruktor yang menginisialisasi akar pohon menjadi null.

1. Function isEmpty():

public boolean isEmpty(): Fungsi untuk memeriksa apakah pohon kosong dengan memeriksa apakah akar pohon root adalah null.

1. Function insert():

public void insert(E data): Fungsi untuk menyisipkan data baru ke dalam pohon biner.

private BTNode2<E> insert(BTNode2<E> node, E data): Fungsi rekursif yang memasukkan data ke dalam pohon. Jika simpul saat ini adalah null, maka simpul baru dengan data tersebut akan dibuat. Jika tidak, fungsi akan mencari tempat yang sesuai untuk menyisipkan data, baik di sisi kiri atau kanan dari simpul saat ini.

1. Function countNodes():

public int countNodes(): Fungsi untuk menghitung jumlah simpul dalam pohon biner.

private int countNodes(BTNode2<E> node): Fungsi rekursif yang menghitung jumlah simpul dalam pohon dengan menghitung jumlah simpul dalam subtree kiri dan kanan serta menambahkan satu untuk simpul saat ini.

1. Function search():

public boolean search(E val): Fungsi untuk mencari elemen tertentu dalam pohon biner.

private boolean search(BTNode2<E> node, E val): Fungsi rekursif yang mencari elemen dalam pohon dengan membandingkan data dari simpul saat ini dengan data yang dicari dan melakukan pencarian rekursif di subtree kiri dan kanan.

1. Traversal Functions (inorder, preorder, postorder):

Fungsi untuk melakukan traversal inorder, preorder, dan postorder pada pohon biner.

Setiap fungsi menerima simpul saat ini sebagai parameter dan melakukan rekursi melalui subtree kiri dan kanan sesuai dengan urutan traversal yang diinginkan, serta mencetak data simpul saat ini.

Implementasi ini menyediakan berbagai operasi umum yang dapat dilakukan pada pohon biner, seperti menyisipkan, mencari, menghitung jumlah simpul, dan melakukan traversal. Ini merupakan dasar untuk membangun struktur data yang lebih kompleks berdasarkan pohon biner.

Class main

import java.util.Scanner;

/\*\*

\* keseluruhan program diambil dari Java Program to Implement Binary Tree

\* https://www.sanfoundry.com/java-program-implement-binary-tree/

\* dengan sedikit perubahan

\*/

public class main {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

/\* Creating object of BT \*/

BinaryTree2 bt = new BinaryTree2();

/\* Perform tree operations \*/

System.out.println("Binary Tree Test\n");

char ch;

do {

System.out.println("\nBinary Tree Operations\n");

System.out.println("1. insert ");

System.out.println("2. search");

System.out.println("3. count nodes");

System.out.println("4. check empty");

int choice = scan.nextInt();

scan.nextLine();

switch (choice) {

case 1:

System.out.println("Enter integer element to insert");

bt.insert(scan.nextLine());

break;

case 2:

System.out.println("Enter integer element to search");

System.out.println("Search result : " + bt.search(scan.nextLine()));

break;

case 3:

System.out.println("Nodes = " + bt.countNodes());

break;

case 4:

System.out.println("Empty status = " + bt.isEmpty());

break;

default:

System.out.println("Wrong Entry \n ");

break;

}

/\* Display tree \*/

System.out.print("\nPost order : ");

bt.postorder();

System.out.print("\nPre order : ");

bt.preorder();

System.out.print("\nIn order : ");

bt.inorder();

System.out.println("\n\nDo you want to continue (Type y or n)\n");

ch = scan.next().charAt(0);

} while (ch == 'Y' || ch == 'y');

}

}

Penjelasan :

Kodingan di atas adalah program Java yang mengimplementasikan penggunaan pohon biner dan berinteraksi dengan pengguna melalui konsol. Program ini memungkinkan pengguna untuk melakukan beberapa operasi pada pohon biner, seperti menyisipkan elemen baru, mencari elemen, menghitung jumlah simpul, dan memeriksa apakah pohon kosong.

Berikut adalah penjelasan singkat dari setiap bagian kodingan tersebut:

1. Import Package:

import java.util.Scanner;: Mengimpor kelas Scanner dari paket java.util untuk membaca input dari pengguna.

1. Main Class (main):

Membuat objek dari kelas Scanner untuk membaca input dari pengguna.

Membuat objek dari kelas BinaryTree2 untuk mewakili pohon biner.

Menampilkan pesan awal yang menandakan bahwa program sedang menguji pohon biner.

Meminta pengguna untuk memilih operasi yang ingin dilakukan pada pohon biner.

Melakukan operasi sesuai dengan pilihan pengguna dan menampilkan hasilnya.

Setelah setiap operasi, menampilkan pohon biner dalam tiga jenis traversal: postorder, preorder, dan inorder.

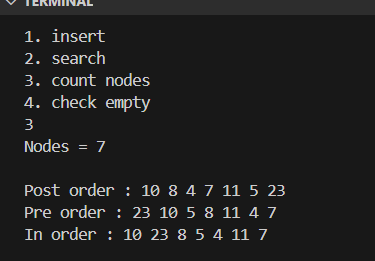
Mengulangi proses ini sampai pengguna memilih untuk berhenti.

1. Interaksi dengan Pengguna:

Setiap operasi yang dapat dilakukan pada pohon biner ditampilkan sebagai opsi yang dapat dipilih oleh pengguna.

Setelah pengguna memilih operasi tertentu, program akan meminta input tambahan jika diperlukan (seperti elemen yang akan disisipkan atau dicari).

Hasil dari operasi yang dipilih akan ditampilkan kepada pengguna.

Program ini memberikan pengalaman interaktif kepada pengguna untuk menggunakan dan memahami operasi dasar yang dapat dilakukan pada pohon biner. Ini juga memberikan tampilan visual dari pohon biner setelah setiap operasi, sehingga pengguna dapat melihat bagaimana pohon berubah sesuai dengan operasi yang dilakukan.  
  
  


A group of white circles with numbers on a black background

Description automatically generated

Inorder : Kiri-Root-Kanan . Pada gambar hasil inorder traversal adalah 10 23 8 5 4 11.

Preorder : Root-Kiri-Kanan . Pada gambar hasil preorder traversal adalah 23 10 5 8 11 4 7.

Preorder : Kanan-Kanan-Root. Pada gambar hasil postorder traversal adalah 10 8 4 2 5 11 7 23