

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №5**

**з дисципліни «Алгоритми та структури даних»**

**на тему: «Дерева»**

Виконав:

студент гр. ПЗ2011 Савенко Я. О.

Прийняла:

Куроп’ятник О. С.

Дніпро, 2022

**Тема.** Дерева.

**Мета.** Ознайомитися з поняттям дерева. Отримати практичні навички реалізації бінарного дерева пошуку.

**Завдання**

Розробити програму мовою java для реалізації бінарного дерево та операцій додавання, видалення, пошуку та перегляду інформації про певний об’єкт (вивід інформаційної частини), симетричного обходу. Клас дерева та вузла представити двома різними класами. Атрибути класу вузла повинні мати модифікатори доступу private або вузол має бути представлений як внутрішній клас класу дерева. Вузол повинен містити посилання на лівий та правий нащадки, поле ключа, поле інформаційної частини, не містити посилання на батьківський вузол. Значення ключа повинно обраховуватися на основі значення інформаційної частини. При додаванні нового вузла зі значенням ключа, яке вже є в дереві, виконати заміну інформаційної частини. Тип даних інформаційної частини визначити за параметризованим типом з індивідуального завдання до лабораторної роботи 4. В програмі реалізувати текстове меню для виконання усіх операцій над деревом в довільному порядку. Для кожного пункту меню передбачити зворотній зв'язок у вигляді результату виконання операції та/або повідомлення про успішне виконання операції.

**Текст програми**

*Файл BinaryTree.java*

**package** com.company;  
  
**public class** BinaryTree<T>  
{  
 **private** Node<T> **root**;  
  
 **private static class** Node<T>  
 {  
 **private int key**;  
 **private** T **data**;  
 **private** Node<T> **left**;  
 **private** Node<T> **right**;  
  
 **public** Node(**int** key, T data)  
 {  
 **this**.**key** = key;  
 **this**.**left** = **null**;  
 **this**.**right** = **null**;  
 **this**.**data** = data;  
 }  
 **private** Node<T> getLeft()  
 {  
 **return left**;  
 }  
 **private** Node<T> getRight()  
 {  
 **return right**;  
 }  
  
 }  
 *// Добавление элемента* **public void** Insert(**int** key, T data)  
 {  
 **root** = doInsert(**root**, key, data);  
 }  
 **private** Node<T> doInsert(Node<T> node, **int** key, T data) {  
 Node<T> newNode = doSearch(node, key);  
 **if**(newNode != **null**)  
 {  
 newNode.**data** = data;  
 **return** node;  
 }  
 **if** (node == **null**)  
 {  
 **return new** Node<T>(key, data);  
 }  
 **else if** (key < node.**key**)  
 {  
 node.**left** = doInsert(node.**left**, key, data);  
 }  
 **else if** (key > node.**key**)  
 {  
 node.**right** = doInsert(node.**right**, key, data);  
 }  
 **return** node;  
 }  
  
 **public void** GetElement(**int** key)  
 {  
 doGetElement(**root**, key);  
 }  
 **private void** doGetElement(Node<T> node, **int** key)  
 {  
 Node<T> newNode = doSearch(node, key);  
 **if**(newNode != **null**)  
 {  
 System.***out***.println(**"Ключ: "** + newNode.**key** + **" Данные: "** + newNode.**data**);  
 }  
 }  
  
 *// поиск ключа* **public boolean** Search(**int** key)  
 {  
 Node<T> node = doSearch(**root**, key);  
 **if**(node != **null**)  
 {  
 **return true**;  
 }  
 **return false**;  
 }  
 **private** Node<T> doSearch(Node<T> node, **int** key) {  
  
 **if** (node != **null**) {  
 **if** (node.**key** == key) {  
 **return** node;  
 } **else if** (key < node.**key**) {  
 **return** doSearch(node.getLeft(), key);  
 } **else if** (key > node.**key**) {  
 **return** doSearch(node.getRight(), key);  
 }  
 }  
 **return null**;  
 }  
  
 *//удаление елемента* **public void** Delete(**int** key)  
 {  
 **root** = doDelete(**root**, key);  
 }  
 **private** Node<T> doDelete(Node<T> current, **int** key)  
 {  
 **if** (current == **null**)  
 {  
 **return null**;  
 }  
 **if** (key == current.**key**)  
 {  
 *//нету детей* **if**(current.**right** == **null** && current.**left** == **null**)  
 {  
 **return null**;  
 }  
  
 *//один потомок* **if**(current.**right** == **null**)  
 {  
 **return** current.**left**;  
 }  
 **if**(current.**left** == **null**)  
 {  
 **return** current.**right**;  
 }  
  
 *//два потомка* Node<T> minNode = minElement(current.**right**);  
 current.**key** = minNode.**key**;  
 current.**data** = minNode.**data**;  
 current.**right** = doDelete(current.**right**, minNode.**key**);  
 **return** current;  
 }  
  
 **if**(key < current.**key**)  
 {  
 current.**left** = doDelete(current.**left**, key);  
 **return** current;  
 }  
 current.**right** = doDelete(current.**right**, key);  
 **return** current;  
 }  
 **private** Node<T> minElement(Node<T> node)  
 {  
 **if**(node.**left** == **null**)  
 {  
 **return** node;  
 }  
 **return** minElement(node.**left**);  
 }  
  
 *//in-order* **public void** InOrder()  
 {  
 doInOrder(**root**);  
 }  
 **private void** doInOrder(Node<T> node)  
 {  
 **if**(node != **null**)  
 {  
 doInOrder(node.**left**);  
 System.***out***.println(**"Ключ: "** + node.**key** + **" Данные: "** + node.**data**);  
 doInOrder(node.**right**);  
 }  
 }  
  
 *//pre-order* **public void** PreOrder()  
 {  
 doPreOrder(**root**);  
 }  
 **private void** doPreOrder(Node<T> node)  
 {  
 **if**(node != **null**)  
 {  
 System.***out***.println(**"Ключ: "** + node.**key** + **" Данные: "** + node.**data**);  
 doPreOrder(node.**left**);  
 doPreOrder(node.**right**);  
 }  
 }  
  
 *//post-order* **public void** PostOrder()  
 {  
 doPostOrder(**root**);  
 }  
 **private void** doPostOrder(Node<T> node)  
 {  
 **if**(node != **null**)  
 {  
 doPostOrder(node.**left**);  
 doPostOrder(node.**right**);  
 System.***out***.println(**"Ключ: "** + node.**key** + **" Данные: "** + node.**data**);  
 }  
 }  
}

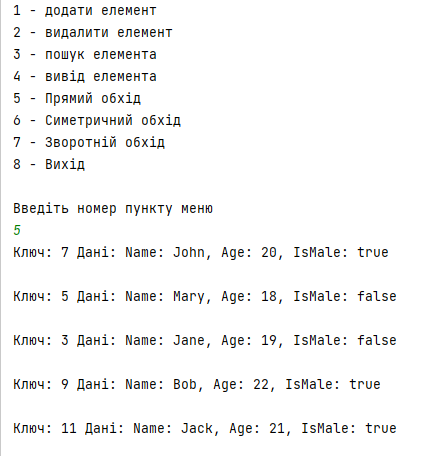
*Файл Patient.java*

**package** com.company;  
  
**public class** Patient  
{  
 **private** String **name**;  
 **private int age**;  
 **private boolean isMale**;  
  
 **public** Patient(String name, **int** age, **boolean** isMale)  
 {  
 **this**.**name** = name;  
 **this**.**age** = age;  
 **this**.**isMale** = isMale;  
 }  
  
 **public** String getName()  
 {  
 **return name**;  
 }  
  
 **public void** setName(String name)  
 {  
 **this**.**name** = name;  
 }  
  
 **public int** getAge()  
 {  
 **return age**;  
 }  
  
 **public void** setAge(**int** age)  
 {  
 **this**.**age** = age;  
 }  
  
 **public boolean** IsMale()  
 {  
 **return isMale**;  
 }  
  
 @Override  
 **public** String toString()  
 {  
 **return "Name: "** + **name** + **", Age: "** + **age** + **", IsMale: "** + **isMale** + **'\n'**;  
 }  
}

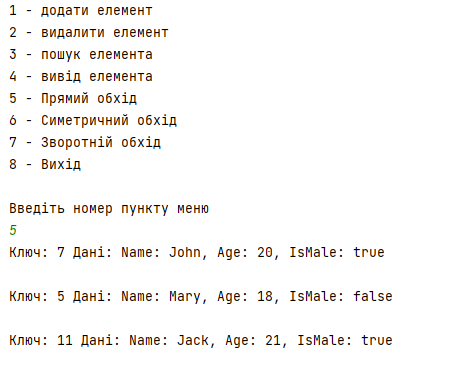
*Файл Main.java*

**package** com.company;  
  
**import** java.util.Scanner;  
  
**public class** Main {  
  
 **public static void** main(String[] args) {  
 BinaryTree<Patient> tree = **new** BinaryTree<Patient>();  
  
 **int** choice = 0;  
 **while** (**true**) {  
 System.***out***.println(**"1 - додати елемент"**);  
 System.***out***.println(**"2 - видалити елемент"**);  
 System.***out***.println(**"3 - пошук елемента"**);  
 System.***out***.println(**"4 - вивід елемента"**);  
 System.***out***.println(**"5 - Прямий обхід"**);  
 System.***out***.println(**"6 - Симетричний обхід"**);  
 System.***out***.println(**"7 - Зворотній обхід"**);  
 System.***out***.println(**"8 - Вихід\n"**);  
 System.***out***.println(**"Введіть номер пункту меню"**);  
 choice = Integer.*parseInt*(**new** Scanner(System.***in***).nextLine());  
 **switch** (choice) {  
 **case** 1:  
 System.***out***.println(**"Введіть ім'я пацієнта"**);  
 String name = **new** Scanner(System.***in***).nextLine();  
 System.***out***.println(**"Введіть вік пацієнта"**);  
 **int** age = Integer.*parseInt*(**new** Scanner(System.***in***).nextLine());  
 System.***out***.println(**"Це чоловік? (true/false)"**);  
 **boolean** isMale = Boolean.*parseBoolean*(**new** Scanner(System.***in***).nextLine());  
 Patient patient = **new** Patient(name, age, isMale);  
 System.***out***.println(**"Введіть ключ для додавання"**);  
 **int** key = Integer.*parseInt*(**new** Scanner(System.***in***).nextLine());  
 tree.Insert(key, patient);  
 **break**;  
 **case** 2:  
 System.***out***.println(**"Введіть ключ для видалення"**);  
 key = Integer.*parseInt*(**new** Scanner(System.***in***).nextLine());  
 tree.Delete(key);  
 **break**;  
 **case** 3:  
 System.***out***.println(**"Введіть ключ для пошуку"**);  
 key = Integer.*parseInt*(**new** Scanner(System.***in***).nextLine());  
 **boolean** result = tree.Search(key);  
 **if** (result) {  
 System.***out***.println(**"Елемент знайдено"**);  
 }  
 **else** {  
 System.***out***.println(**"Елемент не знайдено"**);  
 }  
 **break**;  
 **case** 4:  
 System.***out***.println(**"Введіть ключ для виводу"**);  
 key = Integer.*parseInt*(**new** Scanner(System.***in***).nextLine());  
 tree.GetElement(key);  
 **break**;  
 **case** 5:  
 tree.PreOrder();  
 **break**;  
 **case** 6:  
 tree.InOrder();  
 **break**;  
 **case** 7:  
 tree.PostOrder();  
 **break**;  
 **case** 8:  
 System.***out***.println(**"Вихід з програми"**);  
 System.*exit*(0);  
 **break**;  
 **default**:  
 System.***out***.println(**"Невірний ввід"**);  
 **break**;  
 }  
 }  
 }  
}

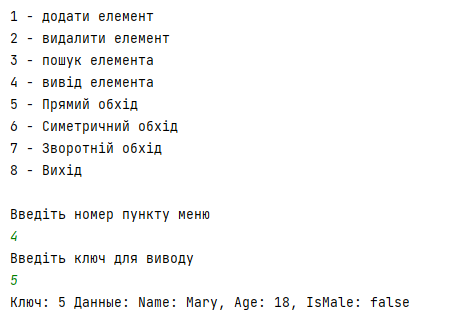
**Результат роботи**



*Рис. 1 – Додані елементи до дерева*

**

*Рис. 2 – Дерево після видалення елементів*

**

*Рис. 3 – Друк елемента по ключу*

**Висновок**

При виконанні лабораторної роботи написав бінарне дерево. Ця структура даних вирішує проблему зв’язного списку з повільним доступом до елементів. Через свою структуру доступ до елементів відбувається за логарифмічний час, проте якщо дерево погано збалансоване, то час може збільшитися. Цю проблему вирішують червоно-чорні та інші види збалансованих дерев.