Міністерство освіти і науки України

КПІ ім. Ігоря Сікорського

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Звіт

З лабораторної роботи №1

“Теорія розробки програмного забезпечення”

Виконала:

студентка ІІІ курсу ФІОТ

групи ІК-12

Петканич Жанна

Київ 2023

Тема роботи: узагальнені типи (Generic) з підтримкою подій. Колекції.

Мета роботи: навчитися проектувати та реалізовувати узагальнені типи, а також типи з підтримкою подій.

Хід роботи

Варіант 3

Завдання

1. Розробити клас власної узагальненої колекції, використовуючи стандартні інтерфейси колекцій із бібліотек System.Collections та System.Collections.Generic. Стандартні колекції при розробці власної не застосовувати. Для колекції передбачити методи внесення даних будь-якого типу, видалення, пошуку та ін. (відповідно до типу колекції).

2. Додати до класу власної узагальненої колекції підтримку подій та обробку виключних ситуацій.

3. Опис класу колекції та всіх необхідних для роботи з колекцією типів зберегти у динамічній бібліотеці.

4. Створити консольний додаток, в якому продемонструвати використання розробленої власної колекції, підписку на події колекції.

Опис класів

Клас BinaryTree<T> представляє власну узагальнену колекцію бінарного дерева. Основні характеристики та функції класу включають:

Властивості та поля:

_root: Посилання на корінь бінарного дерева.

Count: Кількість елементів у дереві.

ItemAdded: Подія, яка спрацьовує при додаванні нового елемента до дерева.

ItemRemoved: Подія, яка спрацьовує при видаленні елемента з дерева.

CollectionCleared: Подія, яка спрацьовує при очищенні всієї колекції.

Конструктори:

BinaryTree(): Конструктор за замовчуванням, ініціалізує порожнє бінарне дерево.

BinaryTree(IComparer<T> comparer): Конструктор, що приймає компаратор для порівняння значень елементів дерева.

Методи:

Add(T item): Додає елемент до дерева та активує подію ItemAdded.

Remove(T item): Видаляє елемент з дерева та активує подію ItemRemoved.

Clear(): Очищує всю колекцію та активує подію CollectionCleared.

CopyTo(T[] array, int arrayIndex): Копіює елементи дерева в масив.

GetEnumerator(): Повертає ітератор для серединного обходу (інфіксного ітератора).

PrintTree(): Виводить структуру бінарного дерева на консоль.

InOrderTraversal(), PreOrderTraversal(), PostOrderTraversal(): Методи для ітеративного обходу дерева в прямому, зворотньому та серединному порядку.

Клас Node<T>:

Клас Node<T> представляє вузол бінарного дерева. Основні характеристики та функції класу включають:

Властивості:

Data: Значення, що зберігається у вузлі.

ParentNode: Посилання на батьківський вузол.

LeftNode, RightNode: Посилання на лівий і правий дочірні вузли.

Конструктор:

Node(T data): Конструктор, що приймає значення інформаційної частини вузла.

Код програми

Клас BinaryTree:

```
public class BinaryTree<T> : ICollection<T>
    where T : IComparable<T>
{
    // Компаратор для порівняння значень
```

```
private readonly IComparer<T> _comparer;

// Корінь
public Node<T> _root;

// Оголошення події для додавання елемента
public event Action<T> ItemAdded;

// Оголошення події для видалення елемента
public event Action<T> ItemRemoved;

// Оголошення події для очищення колекції
public event Action CollectionCleared;



public BinaryTree()
{
   // Ініціалізуємо компаратор за замовчуванням
   _comparer = Comparer<T>.Default;
}

public BinaryTree(IComparer<T> comparer)
{
   _comparer = comparer ?? Comparer<T>.Default;
}

// Кількість елементів у дереві
public int Count { get; private set; }

public bool IsReadOnly => false;

// Додає елемент до дерева
```

```
public void Add(T item)
{
    _root = Insert(_root, item);
    Count++;

    // Відправка сповіщення про подію додавання елемента
    ItemAdded?.Invoke(item);
}

// Рекурсивний метод для вставки елемента
private Node<T> Insert(Node<T> node, T item)
{
    // Якщо вузол порожній
    if (node == null)
    {
        return new Node<T>(item);
    }

    int comparisonResult = _comparer.Compare(item, node.Data);

    // Якщо значення вже існує
    if (comparisonResult == 0)
    {
        return node;
    }
    // Якщо значення менше
    else if (comparisonResult < 0)
    {
        node.LeftNode = Insert(node.LeftNode, item); // Вставляємо вліво
        node.LeftNode.ParentNode = node;
    }
    // Якщо значення більше
```

```
    else
    {
        node.RightNode = Insert(node.RightNode, item); // Вставляємо вправо
        node.RightNode.ParentNode = node;
    }

    return node; // Повертаємо поточний вузол
}

// Видаляє елемент із дерева
public bool Remove(T item)
{
    Node<T> nodeToRemove = FindNode(item);
    if (nodeToRemove != null)
    {
        Remove(nodeToRemove);
        Count--;

        // Відправка сповіщення про подію видалення елемента
        ItemRemoved?.Invoke(item);
        return true;
    }
    return false;
}

// Внутрішній метод для видалення вузла
private void Remove(Node<T> nodeToRemove)
{
    if (nodeToRemove.LeftNode == null)
    {
        // Переміщуємо праве піддерево
        Transplant(nodeToRemove, nodeToRemove.RightNode);
```

```
    }
    else if (nodeToRemove.RightNode == null)
    {
        // Переміщуємо ліве піддерево
        Transplant(nodeToRemove, nodeToRemove.LeftNode);
    }

    // Якщо є обидва піддерева
    else
    {
        // Знаходимо дочірній вузол в правому піддереві
        Node<T> successor = FindMin(nodeToRemove.RightNode);
        if (successor.ParentNode != nodeToRemove)
        {
            // Переміщуємо праве піддерево child
            Transplant(successor, successor.RightNode);
            // Переносимо праве піддерево видаляємого вузла
            successor.RightNode = nodeToRemove.RightNode;
            // Встановлюємо батька для правого піддерева
            successor.RightNode.ParentNode = successor;
        }
        // Переміщуємо child на місце видаленого вузла
        Transplant(nodeToRemove, successor);
        // Переносимо ліве піддерево видаленого вузла
        successor.LeftNode = nodeToRemove.LeftNode;
        // Встановлюємо батька для лівого піддерева
        successor.LeftNode.ParentNode = successor;
    }
}

// Переміщує піддерево
public void Transplant(Node<T> u, Node<T> v)
```

```
{
    //вузол корінь
    if (u.ParentNode == null)
    {
        _root = v;
    }

    // Якщо вузол лівий дочірній вузол
    else if (u == u.ParentNode.LeftNode)
    {
        u.ParentNode.LeftNode = v;
    }

    // Якщо вузол правий дочірній вузол
    else
    {
        u.ParentNode.RightNode = v;
    }

    if (v != null)
    {
        v.ParentNode = u.ParentNode;
    }
}

// Знаходить вузол з мінімальним значенням
private static Node<T> FindMin(Node<T> node)
{
    while (node.LeftNode != null)
    {
        node = node.LeftNode;
    }
```

```
    return node;
}

// Перевіряє, чи міститься елемент у дереві
public bool Contains(T item)
{
    return FindNode(item) != null;
}

// Знаходить вузол за значенням
public Node<T> FindNode(T item)
{
    Node<T> currentNode = _root;
    while (currentNode != null)
    {
        int comparisonResult = _comparer.Compare(item, currentNode.Data);
        if (comparisonResult == 0)
        {
            return currentNode;
        }
        else if (comparisonResult < 0)
        {
            currentNode = currentNode.LeftNode;
        }
        else
        {
            currentNode = currentNode.RightNode;
        }
    }
    return null;
}
```

```
// Очищує дерево
public void Clear()
{
    _root = null;
    Count = 0; // Скидаємо лічильник

    // Відправка сповіщення про очищення колекції
    CollectionCleared?.Invoke();
}

// Копіює елементи в масив
public void CopyTo(T[] array, int arrayIndex)
{
    foreach (var item in InOrderTraversal())
    {
        array[arrayIndex++] = item;
    }
}

public IEnumerator<T> GetEnumerator()
{
    return InOrderTraversal().GetEnumerator();
}

// Виводить структуру дерева на консоль
public void PrintTree()
{
    PrintTree(_root, string.Empty);
}

// Приватний метод для виводу структури дерева
private void PrintTree(Node<T> node, string indent)
```

```
        {
            if (node != null)
            {
                Console.WriteLine($"{indent}{node.Data}");
                if (node.LeftNode != null || node.RightNode != null)
                {
                    PrintTree(node.LeftNode, indent + "  |");
                    PrintTree(node.RightNode, indent + "    ");
                }
            }
        }

        public IEnumerable<T> InOrderTraversal()
        {
            return InOrderTraversal(_root);
        }

        //Метод для серединного обходу
        private IEnumerable<T> InOrderTraversal(Node<T> node)
        {
            if (node != null) // Якщо вузол не порожній
            {
                foreach (var leftNodeData in InOrderTraversal(node.LeftNode)) // Рекурсивно обходимо ліве піддерево
                {
                    yield return leftNodeData;
                }
                yield return node.Data; // Повертаємо значення поточного вузла
                foreach (var rightNodeData in InOrderTraversal(node.RightNode)) // Рекурсивно обходимо праве піддерево
                {
                    yield return rightNodeData;
                }
```

```
        }
    }
    public IEnumerable<T> PreOrderTraversal()
    {
        return PreOrderTraversal(_root);
    }

    // Метод для прямого обходу
    private IEnumerable<T> PreOrderTraversal(Node<T> node)
    {
        if (node != null) // Якщо вузол не порожній
        {
            yield return node.Data; // Повертаємо значення поточного вузла
            foreach (var leftNodeData in PreOrderTraversal(node.LeftNode)) // Рекурсивно обходимо ліве піддерево
            {
                yield return leftNodeData;
            }
            foreach (var rightNodeData in PreOrderTraversal(node.RightNode)) // Рекурсивно обходимо праве піддерево
            {
                yield return rightNodeData;
            }
        }
    }

    public IEnumerable<T> PostOrderTraversal()
    {
        return PostOrderTraversal(_root);
    }

    // Метод для зворотнього обходу
    private IEnumerable<T> PostOrderTraversal(Node<T> node)
```

```
    {
        if (node != null)
        {
            foreach (var leftNodeData in PostOrderTraversal(node.LeftNode)) // Рекурсивно обходимо ліве піддерево
            {
                yield return leftNodeData;
            }
            foreach (var rightNodeData in PostOrderTraversal(node.RightNode)) // Рекурсивно обходимо праве піддерево
            {
                yield return rightNodeData;
            }
            yield return node.Data; // Повертаємо значення поточного вузла
        }
    }

    System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
    {
        throw new NotImplementedException();
    }
```

Клас Node:

```
public class Node<T>
  {
    public T Data { get; set; }
    public Node<T> ParentNode { get; set; }
    public Node<T> LeftNode { get; set; }
    public Node<T> RightNode { get; set; }

    public Node(T data)
    {
        Data = data;
```

```
    }
  }
```

Клас Program:

```
class Program
  {
    static void Main()
    {
      BinaryTree<int> binaryTree = new BinaryTree<int>();

      while (true)
      {
        Console.WriteLine("Виберіть:");
        Console.WriteLine("1. Додати елемент");
        Console.WriteLine("2. Видалити елемент");
        Console.WriteLine("3. Пошук елемента");
        Console.WriteLine("4. Вивести бінарне дерево");
        Console.WriteLine("5. Вийти");

        string choice = Console.ReadLine();

        switch (choice)
        {
          case "1":
            Console.Write("Введіть ціле число для додавання: ");
            if (int.TryParse(Console.ReadLine(), out int addValue))
            {
              binaryTree.Add(addValue);
              Console.WriteLine($"Елемент {addValue} додано до бінарного дерева.");
            }
            else
            {
```

```
            Console.WriteLine("Некоректний формат числа.");
        }
        break;

    case "2":
        Console.Write("Введіть ціле число для видалення: ");
        if (int.TryParse(Console.ReadLine(), out int removeValue))
        {
            if (binaryTree.Remove(removeValue))
            {
                Console.WriteLine($"Елемент {removeValue} видалено з бінарного дерева.");
            }
            else
            {
                Console.WriteLine($"Елемент {removeValue} не знайдено в бінарному дереві.");
            }
        }
        else
        {
            Console.WriteLine("Некоректний формат числа.");
        }
        break;

    case "3":
        Console.Write("Введіть ціле число для пошуку: ");
        if (int.TryParse(Console.ReadLine(), out int searchValue))
        {
            if (binaryTree.Contains(searchValue))
            {
                Console.WriteLine($"Елемент {searchValue} знайдено в бінарному дереві.");
            }
            else
```

```
                {
                    Console.WriteLine($"Елемент {searchValue} не знайдено в бінарному дереві.");
                }
            }
            else
            {
                Console.WriteLine("Некоректний формат числа.");
            }
            break;

        case "4":
            Console.WriteLine("Бінарне дерево:");
            binaryTree.PrintTree();
            break;

        case "5":
            Environment.Exit(0);
            break;

        default:
            Console.WriteLine("Некоректний вибір. Спробуйте ще раз.");
            break;
    }
  }
 }
}
```