**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет технологий»

Отчет по лабораторной работе №3

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-33Б |  | преподаватель каф. ИУ5 |
| Кузнецов Григорий |  |  |
| Подпись и дата: |  | Подпись и дата: |

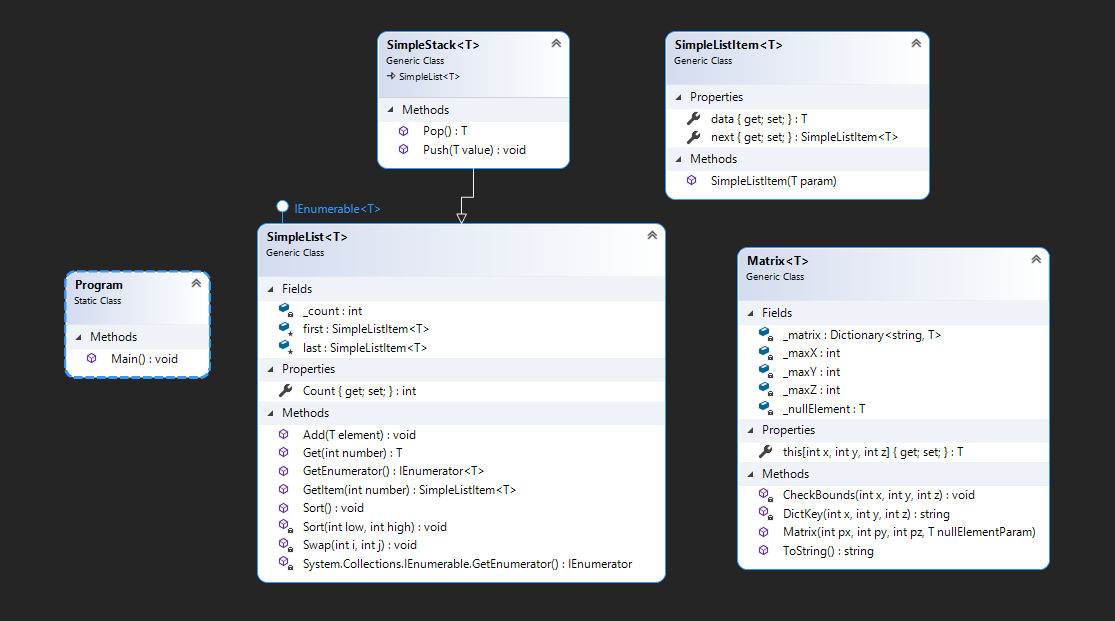
Москва, 2020 г.

**Описание задания**

Разработать программу, реализующую работу с коллекциями.

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создать объекты классов «Прямоугольник», «Квадрат», «Круг».
3. Для реализации возможности сортировки геометрических фигур для класса «Геометрическая фигура» добавить реализацию интерфейса IComparable. Сортировка производится по площади фигуры.
4. Создать коллекцию класса ArrayList. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.
5. Создать коллекцию класса List<Figure>. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.
6. Модифицировать класс разреженной матрицы (проект SparseMatrix) для работы с тремя измерениями – x,y,z. Вывод элементов в методе ToString() осуществлять в том виде, который Вы считаете наиболее удобным. Разработать пример использования разреженной матрицы для геометрических фигур.
7. Реализовать класс «SimpleStack» на основе односвязного списка. Класс SimpleStack наследуется от класса SimpleList (проект SimpleListProject). Необходимо добавить в класс методы:
   * public void Push(T element) – добавление в стек;
   * public T Pop() – чтение с удалением из стека.
8. Пример работы класса SimpleStack реализовать на основе геометрических фигур.

**Диаграмма классов**

****

**Текст программы**

Matrix.cs:

using System;

using System.Collections.Generic;

using System.Text;

namespace SparseMatrix

{

public class Matrix<T>

{

/// <summary>

/// Словарь для хранения значений

/// </summary>

private readonly Dictionary<string, T> \_matrix = new Dictionary<string, T>();

/// <summary>

/// Количество элементов по горизонтали (максимальное количество столбцов)

/// </summary>

private readonly int \_maxX;

/// <summary>

/// Количество элементов по вертикали (максимальное количество строк)

/// </summary>

private readonly int \_maxY;

private readonly int \_maxZ;

/// <summary>

/// Пустой элемент, который возвращается если элемент с нужными координатами не был задан

/// </summary>

private readonly T \_nullElement;

/// <summary>

/// Конструктор

/// </summary>

public Matrix(int px, int py, int pz, T nullElementParam)

{

\_maxX = px;

\_maxY = py;

\_maxZ = pz;

\_nullElement = nullElementParam;

}

/// <summary>

/// Индексатор для доступа к данных

/// </summary>

public T this[int x, int y, int z]

{

get

{

CheckBounds(x, y, z);

var key = DictKey(x, y, z);

if (\_matrix.ContainsKey(key))

{

return \_matrix[key];

}

else

{

return \_nullElement;

}

}

set

{

CheckBounds(x, y, z);

var key = DictKey(x, y, z);

\_matrix.Add(key, value);

}

}

/// <summary>

/// Проверка границ

/// </summary>

private void CheckBounds(int x, int y, int z)

{

if (x < 0 || x >= \_maxX) throw new Exception("x=" + x + " выходит за границы");

if (y < 0 || y >= \_maxY) throw new Exception("y=" + y + " выходит за границы");

if (z < 0 || z >= \_maxZ) throw new Exception("z=" + z + " выходит за границы");

}

/// <summary>

/// Формирование ключа

/// </summary>

private string DictKey(int x, int y, int z) => x.ToString() + "\_" + y.ToString() + "\_" + z.ToString();

/// <summary>

/// Приведение к строке

/// </summary>

/// <returns></returns>

public override string ToString()

{

//Класс StringBuilder используется для построения длинных строк

//Это увеличивает производительность по сравнению с созданием и склеиванием

//большого количества обычных строк

var b = new StringBuilder();

for (var z = 0; z < \_maxZ; z++)

{

for (var j = 0; j < \_maxY; j++)

{

b.Append("[");

for (var i = 0; i < \_maxX; i++)

{

if (i > 0) b.Append("\t");

b.Append(this[i, j, z]?.ToString() ?? "null");

}

b.Append("]\n");

}

b.Append('\n');

}

return b.ToString();

}

}

}

SimpleList.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace SimpleListProject

{

/// <summary>

/// Элемент списка

/// </summary>

public class SimpleListItem<T>

{

/// <summary>

/// Данные

/// </summary>

public T data { get; set; }

/// <summary>

/// Следующий элемент

/// </summary>

public SimpleListItem<T> next { get; set; }

///конструктор

public SimpleListItem(T param)

{

this.data = param;

}

}

/// <summary>

/// Список

/// </summary>

public class SimpleList<T> : IEnumerable<T>

where T : IComparable

{

/// <summary>

/// Первый элемент списка

/// </summary>

protected SimpleListItem<T> first = null;

/// <summary>

/// Последний элемент списка

/// </summary>

protected SimpleListItem<T> last = null;

/// <summary>

/// Количество элементов

/// </summary>

public int Count

{

get { return \_count; }

protected set { \_count = value; }

}

int \_count;

/// <summary>

/// Добавление элемента

/// </summary>

/// <param name="element"></param>

public void Add(T element)

{

SimpleListItem<T> newItem = new SimpleListItem<T>(element);

this.Count++;

//Добавление первого элемента

if (last == null)

{

this.first = newItem;

this.last = newItem;

}

//Добавление следующих элементов

else

{

//Присоединение элемента к цепочке

this.last.next = newItem;

//Просоединенный элемент считается последним

this.last = newItem;

}

}

/// <summary>

/// Чтение контейнера с заданным номером

/// </summary>

public SimpleListItem<T> GetItem(int number)

{

if ((number < 0) || (number >= this.Count))

{

//Можно создать собственный класс исключения

throw new Exception("Выход за границу индекса");

}

SimpleListItem<T> current = this.first;

int i = 0;

//Пропускаем нужное количество элементов

while (i < number)

{

//Переход к следующему элементу

current = current.next;

//Увеличение счетчика

i++;

}

return current;

}

/// <summary>

/// Чтение элемента с заданным номером

/// </summary>

public T Get(int number)

{

return GetItem(number).data;

}

/// <summary>

/// Для перебора коллекции

/// </summary>

public IEnumerator<T> GetEnumerator()

{

SimpleListItem<T> current = this.first;

//Перебор элементов

while (current != null)

{

//Возврат текущего значения

yield return current.data;

//Переход к следующему элементу

current = current.next;

}

}

//Реализация обощенного IEnumerator<T> требует реализации необобщенного интерфейса

//Данный метод добавляется автоматически при реализации интерфейса

System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

/// <summary>

/// Cортировка

/// </summary>

public void Sort()

{

Sort(0, this.Count - 1);

}

/// <summary>

/// Алгоритм быстрой сортировки

/// </summary>

/// <param name="low"></param>

/// <param name="high"></param>

private void Sort(int low, int high)

{

int i = low;

int j = high;

T x = Get((low + high) / 2);

do

{

while (Get(i).CompareTo(x) < 0) ++i;

while (Get(j).CompareTo(x) > 0) --j;

if (i <= j)

{

Swap(i, j);

i++; j--;

}

} while (i <= j);

if (low < j) Sort(low, j);

if (i < high) Sort(i, high);

}

/// <summary>

/// Вспомогательный метод для обмена элементов при сортировке

/// </summary>

private void Swap(int i, int j)

{

SimpleListItem<T> ci = GetItem(i);

SimpleListItem<T> cj = GetItem(j);

T temp = ci.data;

ci.data = cj.data;

cj.data = temp;

}

}

}

SimpleStack.cs:

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab3

{

public class SimpleStack<T> : SimpleListProject.SimpleList<T> where T : IComparable

{

public void Push(T value) =>

Add(value);

public T Pop()

{

var ret = last;

if (Count - 2 >= 0)

{

var secondToLast = GetItem(Count - 2);

secondToLast.next = null;

last = secondToLast;

}

Count--;

return ret.data;

}

}

}

Program.cs:

using System;

using Lab2;

using System.Collections;

using System.Collections.Generic;

namespace Lab3

{

public static class Program

{

public static void Main()

{

var square = new Square(2d);

var rectangle = new Rectangle(1d, 2d);

var circle = new Circle(1d);

var array = new ArrayList

{

square,

rectangle,

circle

};

array.Sort();

foreach (Figure figure in array)

figure.Print();

var list = new List<Figure>

{

square,

rectangle,

circle

};

list.Sort();

foreach (var figure in list)

figure.Print();

var matrix = new SparseMatrix.Matrix<Figure>(3, 3, 3, null);

matrix[0, 0, 0] = square;

matrix[0, 1, 0] = rectangle;

matrix[0, 0, 1] = circle;

Console.WriteLine(matrix);

var stack = new SimpleStack<Figure>();

stack.Push(square);

stack.Push(circle);

stack.Push(rectangle);

stack.Pop().Print();

stack.Pop().Print();

stack.Pop().Print();

}

}

}

**Пример выполнения программы**

