|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

**Факультет «Радиотехнический»**

**Кафедра «Системы обработки информации и управления»**

Лабораторная работа № 3

по дисциплине «Базовые компоненты интернет технологий»

Выполнил:

студент(ка) группы № РТ5-31Б

Пакало Александр Сергеевич

подпись, дата

Проверил:

преподаватель каф. ИУ5  
Гапанюк Юрий Евгеньевич

подпись, дата

2020 г.

Оглавление

[Описание задания. 3](#_Toc59481073)

[Диаграмма классов. 4](#_Toc59481074)

[Текст программы. 4](#_Toc59481075)

[GeometricFigure.cs 4](#_Toc59481076)

[GeometricFigureMatrixCheckEmpty.cs 5](#_Toc59481077)

[SimpleList.cs 5](#_Toc59481078)

[SimpleListItem.cs 7](#_Toc59481079)

[SimpleStack.cs 8](#_Toc59481080)

[SparseMatrix.cs 9](#_Toc59481081)

[Анализ результатов. 12](#_Toc59481082)

# Описание задания.

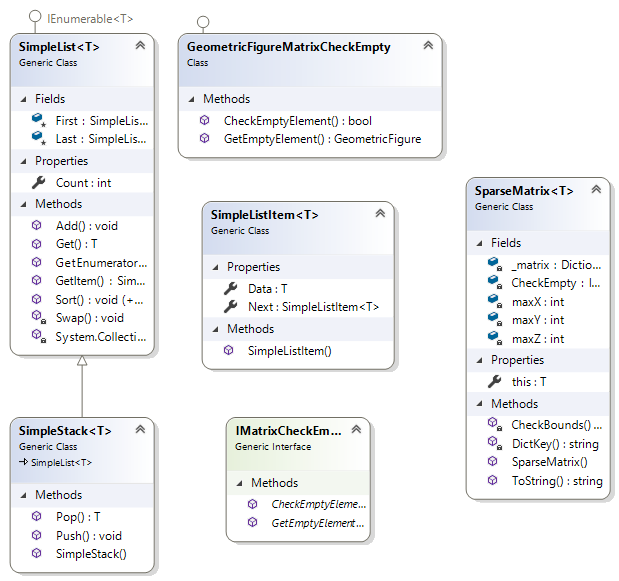
Разработать программу, реализующую работу с коллекциями.

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.

**Примечание:**

Используемые элементы из второй лабораторной работы не будут здесь повторно расписаны с целью предотвращения дублирования отчёта второй лабораторной работы. Единственное исключение – класс Geometric Figure, в которой добавлена реализация интерфейса IComparable.

# Диаграмма классов.



# Текст программы.

## GeometricFigure.cs

using System;

namespace GeometricFigures

{

public abstract class GeometricFigure : IPrint, IComparable

{

public string FigureType { get; protected set; }

public abstract double Area();

public int CompareTo(object obj)

{

// Приведение параметра к типу "фигура".

GeometricFigure p = (GeometricFigure)obj;

// Сравнение.

if (this.Area() < p.Area()) return -1;

else if (this.Area() == p.Area()) return 0;

else return 1; //(this.Area() > p.Area())

}

public override string ToString()

{

return this.FigureType.ToString() + " Area = " + Area();

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

## GeometricFigureMatrixCheckEmpty.cs

using GeometricFigures;

using CustomCollections;

class GeometricFigureMatrixCheckEmpty : IMatrixCheckEmpty<GeometricFigure>

{

/// <summary>

/// В качестве пустого элемента возвращается null

/// </summary>

public GeometricFigure GetEmptyElement()

{

return null;

}

/// <summary>

/// Проверка что переданный параметр равен null

/// </summary>

public bool CheckEmptyElement(GeometricFigure element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

}

## SimpleList.cs

using System;

using System.Collections.Generic;

namespace CustomCollections

{

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; protected set; }

/// <summary>

/// Добавление элемента.

/// </summary>

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;

//Присоединенный 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 ArgumentOutOfRangeException("Выход за границу индекса");

}

SimpleListItem<T> Current = this.First;

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

{

Current = Current.Next;

}

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 обобщенного 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>

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;

}

}

}

## SimpleListItem.cs

using System;

namespace CustomCollections

{

/// <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;

}

}

}

## SimpleStack.cs

using System;

using System.Collections;

using System.Linq.Expressions;

using System.Text;

namespace CustomCollections

{

/// <summary>

/// Класс стек.

/// </summary>

class SimpleStack<T> : SimpleList<T> where T : IComparable

{

public SimpleStack(params T[] elements)

{

foreach (T Element in elements)

{

Push(Element);

}

}

/// <summary>

/// Добавление в стек

/// </summary>

public void Push(T Element)

{

//Добавление в конец списка уже реализовано.

Add(Element);

}

/// <summary>

/// Удаление и чтение из стека.

/// </summary>

public T Pop()

{

// default(T) - значение для типа T по умолчанию.

T Result = default(T);

// Если стек пуст, возвращается значение по умолчанию для типа.

if (this.Count == 0) return Result;

// Если элемент единственный.

if (this.Count == 1)

{

// То из него читаются данные.

Result = this.First.Data;

// Обнуляются указатели начала и конца списка.

this.First = null;

this.Last = null;

}

// В списке более одного элемента.

else

{

// Поиск предпоследнего элемента.

SimpleListItem<T> newLast = this.GetItem(this.Count - 2);

// Чтение значения из последнего элемента.

Result = newLast.Next.Data;

// Предпоследний элемент считается последним.

this.Last = newLast;

// Последний элемент удаляется из списка.

newLast.Next = null;

}

// Уменьшение количества элементов в списке.

this.Count--;

// Возврат результата.

return Result;

}

}

}

## SparseMatrix.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CustomCollections

{

/// <summary>

/// Проверка пустого элемента матрицы

/// </summary>

public interface IMatrixCheckEmpty<T>

{

/// <summary>

/// Возвращает пустой элемент

/// </summary>

T GetEmptyElement();

/// <summary>

/// Проверка что элемент является пустым

/// </summary>

bool CheckEmptyElement(T element);

}

public class SparseMatrix<T>

{

/// <summary>

/// Словарь для хранения значений

/// </summary>

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

/// <summary>

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

/// </summary>

readonly int maxX;

/// <summary>

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

/// </summary>

readonly int maxY;

/// <summary>

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

/// </summary>

readonly int maxZ;

/// <summary>

/// Реализация интерфейса для проверки пустого элемента

/// </summary>

readonly IMatrixCheckEmpty<T> CheckEmpty;

/// <summary>

/// Конструктор

/// </summary>

public SparseMatrix(int px, int py, int pz,

IMatrixCheckEmpty<T> CheckEmptyParam)

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.CheckEmpty = CheckEmptyParam;

}

/// <summary>

/// Индексатор для доступа к данных

/// </summary>

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

{

set

{

CheckBounds(x, y, z);

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

this.\_matrix.Add(key, value);

}

get

{

CheckBounds(x, y, z);

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

if (this.\_matrix.ContainsKey(key))

{

return this.\_matrix[key];

}

else

{

return this.CheckEmpty.GetEmptyElement();

}

}

}

/// <summary>

/// Проверка границ

/// </summary>

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

{

if (x < 0 || x >= this.maxX)

{

throw new ArgumentOutOfRangeException("x",

"x=" + x + " выходит за границы");

}

if (y < 0 || y >= this.maxY)

{

throw new ArgumentOutOfRangeException("y",

"y=" + y + " выходит за границы");

}

if (z < 0 || z >= this.maxZ)

{

throw new ArgumentOutOfRangeException("z",

"z=" + z + " выходит за границы");

}

}

/// <summary>

/// Формирование ключа

/// </summary>

string DictKey(int x, int y, int z)

{

return x.ToString() + "\_" + y.ToString() + "\_" + z.ToString();

}

/// <summary>

/// Приведение к строке

/// </summary>

/// <returns></returns>

public override string ToString()

{

StringBuilder b = new StringBuilder();

for (int k = 0; k < this.maxZ; k++)

{

b.Append("[");

for (int j = 0; j < this.maxY; j++)

{

// Добавление разделителя-табуляции.

if (j > 0)

{

b.Append(" ");

}

b.Append("[ ");

for (int i = 0; i < this.maxX; i++)

{

// Добавление разделителя-табуляции.

if (i > 0)

{

b.Append("; ");

}

// Если текущий элемент не пустой.

if (!this.CheckEmpty.CheckEmptyElement(this[i, j, k]))

{

// Добавить приведенный к строке текущий элемент.

b.Append(this[i, j, k].ToString());

}

else

{

// Иначе добавить признак пустого значения.

b.Append("- ");

}

}

b.Append("]");

}

b.Append("]\n");

}

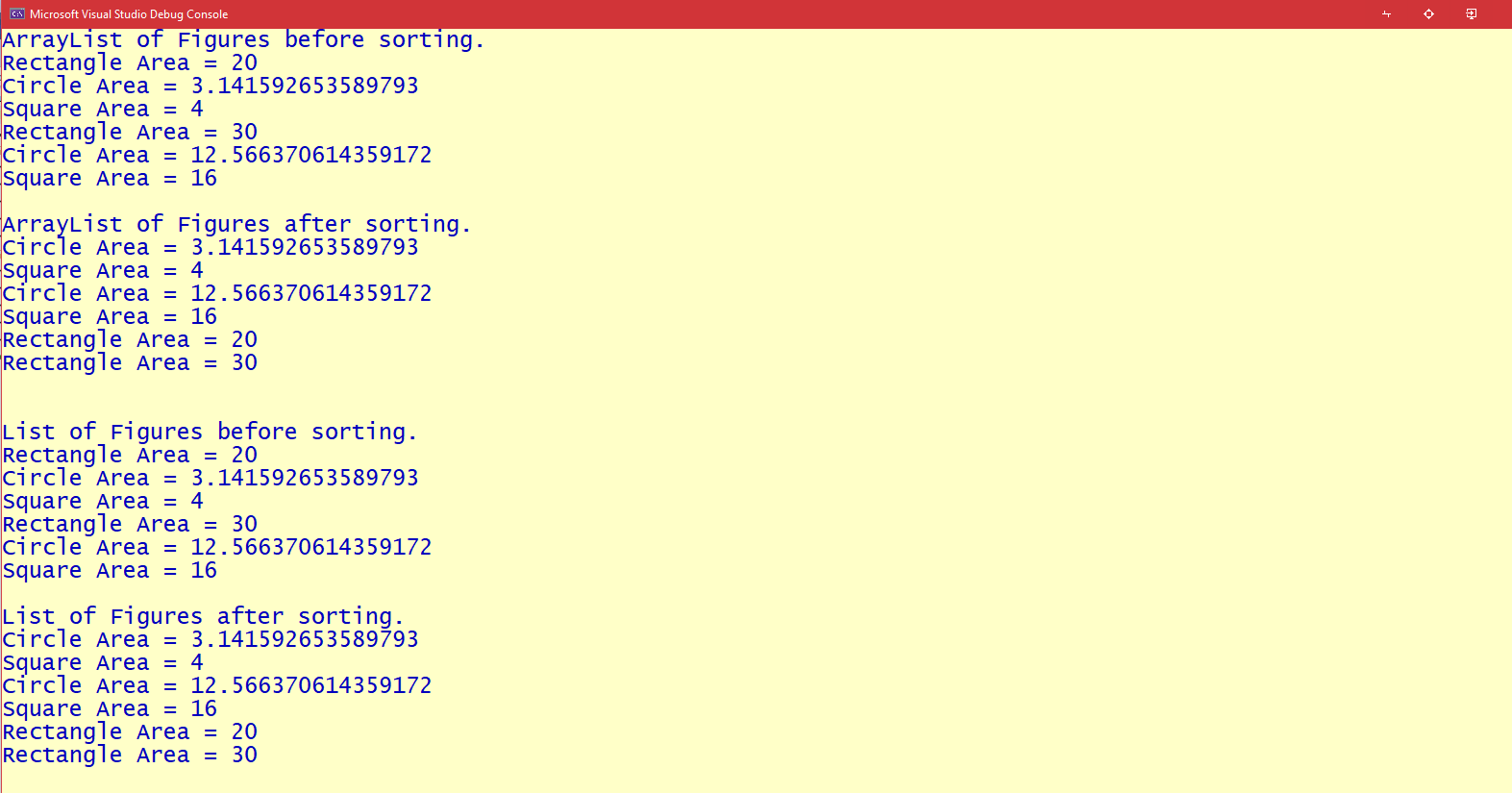
return b.ToString();

}

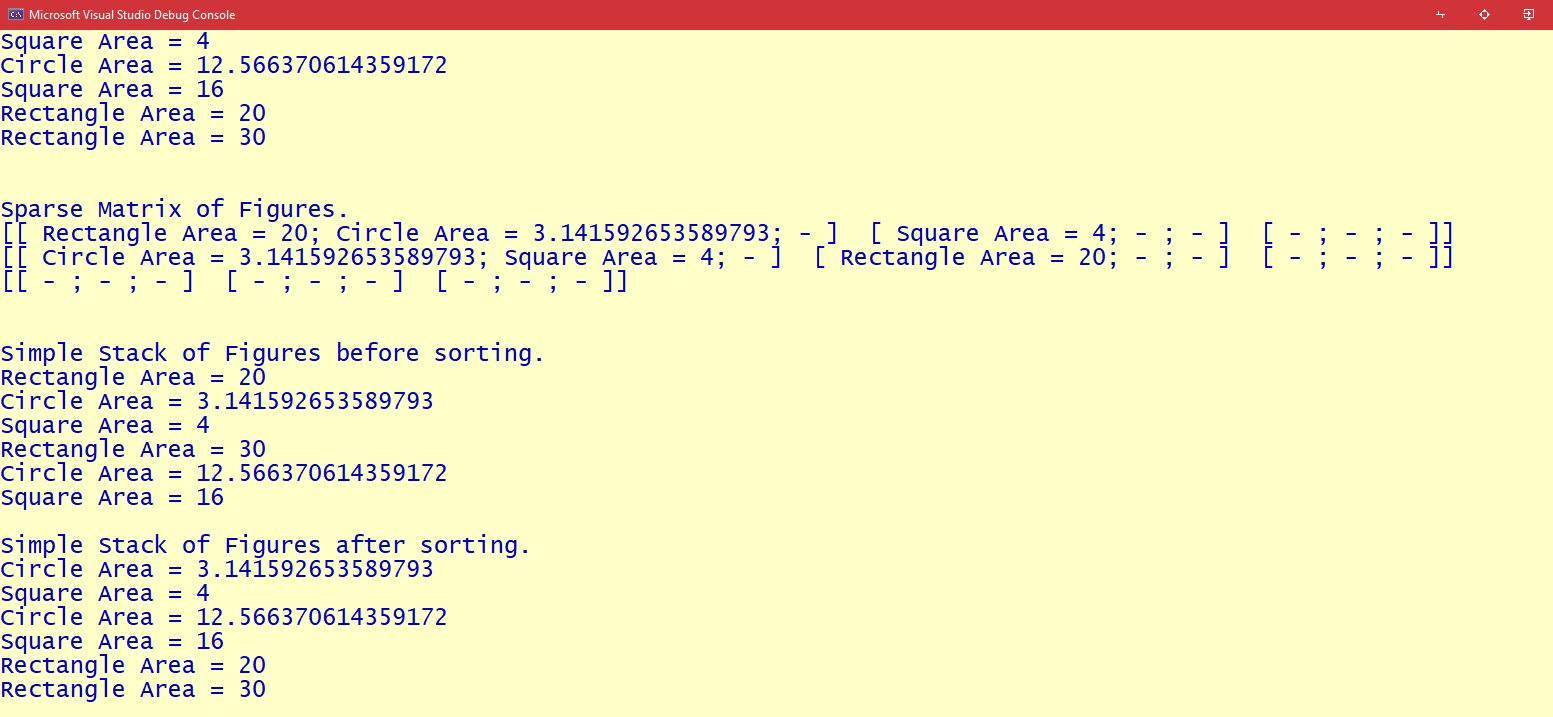
}

}

# Анализ результатов.



Скриншот 1 (демонстрация работы программы).



Скриншот 2 (демонстрация работы программы).