Московский государственный технический университет

им. Н. Э. Баумана

**Отчет**

**по лабораторной работе №3**

Выполнил:

Студент группы ИУ5-35Б

Богданова В.В.

Проверил:

Преподаватель

Гапанюк Ю.Е.

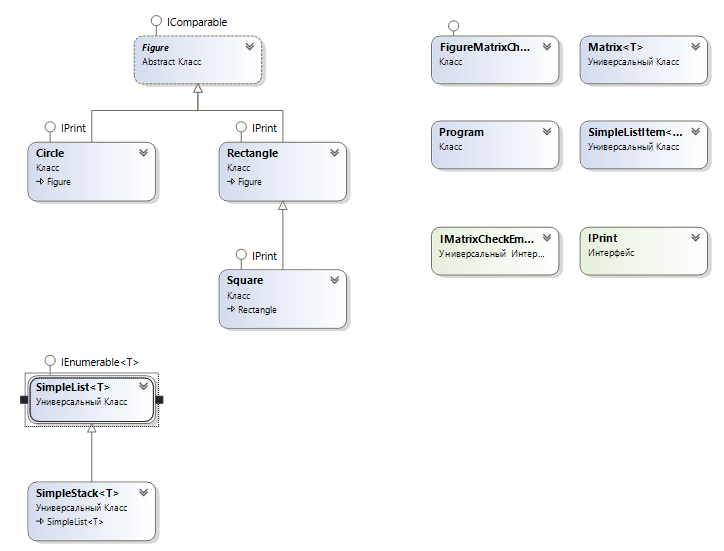
2020г.

**1). 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.

**2). Диаграмма классов**



**3). Текст программы**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Matrix

{

/// <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 Matrix<T>

{

/// <summary>

/// Словарь для хранения значений

/// </summary>

Dictionary<string, T> matrix = new Dictionary<string, T>();

/// <summary>

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

/// </summary>

int maxX;

/// <summary>

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

/// </summary>

int maxY;

/// <summary>

/// Реализация интерфейса для проверки пустого элемента

/// </summary>

IMatrixCheckEmpty<T> сheckEmpty;

/// <summary>

/// Конструктор

/// </summary>

public Matrix(int x, int y, IMatrixCheckEmpty<T> сheckEmptyParam)

{

this.maxX = x;

this.maxY = y;

this.сheckEmpty = сheckEmptyParam;

}

/// <summary>

/// Индексатор для доступа к данных

/// </summary>

public T this[int x, int y]

{

set

{

string key = DictKey(x, y);

this.matrix.Add(key, value);

}

get

{

string key = DictKey(x, y);

if (this.matrix.ContainsKey(key))

{

return this.matrix[key];

}

else

{

return this.сheckEmpty.getEmptyElement();

}

}

}

/// <summary>

/// Формирование ключа

/// </summary>

string DictKey(int x, int y)

{

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

}

/// <summary>

/// Приведение к строке

/// </summary>

public override string ToString()

{

StringBuilder b = new StringBuilder();

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

{

b.Append("[");

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

{

//Добавление разделителя-табуляции

if (i > 0)

{

b.Append("\t");

}

//Если текущий элемент не пустой

if (!this.сheckEmpty.checkEmptyElement(this[i, j]))

{

//Добавить приведенный к строке текущий элемент

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

}

else

{

//Иначе добавить признак пустого значения

b.Append(" 0 ");

}

}

b.Append("]\n");

}

return b.ToString();

}

}

}

namespace SimpleListStack

{

/// <summary>

/// Список

/// </summary>

public class SimpleList<T> : IEnumerable<T> //для перебора элементов

where T : IComparable //для сравнения

{

/// <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>

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;

}

}

//Необобщенный интерфейс, добавляется автоматически к IEnumerator<T>

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;

}

}

/// <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; }

/// <summary>

///конструктор

/// </summary>

public SimpleListItem(T param)

{

this.data = param;

}

}

/// <summary>

/// Класс стек

/// </summary>

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

{

/// <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;

}

}

}

namespace Figures

{

/// <summary>

/// Класс фигура

/// </summary>

abstract class Figure : IComparable

{

/// <summary>

/// Тип фигуры

/// </summary>

public string Type { get; protected set; }

/// <summary>

/// Вычисление площади

/// </summary>

public abstract double Area();

/// <summary>

/// Приведение к строке, переопределение метода Object

/// </summary>

public override string ToString()

{

return this.Type + " площадью " +

this.Area().ToString();

}

/// <summary>

/// Реализация интерфейса IComparable для использования метода Sort()

/// </summary>

public int CompareTo(object o)

{

Figure fig = o as Figure;

return this.Area().CompareTo(fig.Area());

}

}

/// <summary>

/// Интерфейс для печати

/// </summary>

interface IPrint

{

void Print();

}

/// <summary>

/// Прямоугольник

/// </summary>

class Rectangle : Figure, IPrint

{

/// <summary>

/// Высота

/// </summary>

double height;

/// <summary>

/// Ширина

/// </summary>

double width;

/// <summary>

/// Основной конструктор

/// </summary>

/// <param name="ph">Высота</param>

/// <param name="pw">Ширина</param>

public Rectangle(double ph, double pw)

{

this.height = ph;

this.width = pw;

this.Type = "Прямоугольник";

}

/// <summary>

/// Вычисление площади

/// </summary>

public override double Area()

{

double Result = this.width \* this.height;

return Result;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

/// <summary>

/// Квадрат

/// </summary>

class Square : Rectangle, IPrint

{

public Square(double size)

: base(size, size)

{

this.Type = "Квадрат";

}

}

/// <summary>

/// Класс круг

/// </summary>

class Circle : Figure, IPrint

{

/// <summary>

/// Ширина

/// </summary>

double radius;

/// <summary>

/// Основной конструктор

/// </summary>

/// <param name="ph">Высота</param>

/// <param name="pw">Ширина</param>

public Circle(double pr)

{

this.radius = pr;

this.Type = "Круг";

}

public override double Area()

{

double Result = Math.PI \* this.radius \* this.radius;

return Result;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

class FigureMatrixCheckEmpty : Matrix.IMatrixCheckEmpty<Figure>

{

/// <summary>

/// В качестве пустого элемента возвращается null

/// </summary>

public Figure getEmptyElement()

{

return null;

}

/// <summary>

/// Проверка что переданный параметр равен null

/// </summary>

public bool checkEmptyElement(Figure element)

{

bool emp = false;

if (element == null)

{

emp = true;

}

return emp;

}

}

}

namespace Program

{

class Program

{

static void Main(string[] args)

{

Figures.Rectangle rect = new Figures.Rectangle(5, 4);

Figures.Square square = new Figures.Square(5);

Figures.Circle circle = new Figures.Circle(5);

Console.WriteLine("Коллекция ArrayList\n");

ArrayList figures = new ArrayList() { square, circle, rect };

Console.WriteLine("До сортировки:");

foreach (var x in figures) Console.WriteLine(x);

figures.Sort();

Console.WriteLine("\nПосле сортировки:");

foreach (var x in figures) Console.WriteLine(x);

Console.WriteLine("\nКоллекция List\n");

List<Figures.Figure> figures\_list = new List<Figures.Figure>() { square, circle, rect };

Console.WriteLine("До сортировки:");

foreach (var x in figures\_list) Console.WriteLine(x);

figures\_list.Sort();

Console.WriteLine("\nПосле сортировки:");

foreach (var x in figures\_list) Console.WriteLine(x);

Console.WriteLine("\nМатрица");

Matrix.Matrix<Figures.Figure> matrix = new Matrix.Matrix<Figures.Figure>(3, 3, new Figures.FigureMatrixCheckEmpty());

matrix[0, 0] = rect;

matrix[1, 1] = square;

matrix[2, 2] = circle;

Console.WriteLine(matrix.ToString());

Console.WriteLine("\nСписок");

SimpleListStack.SimpleList<Figures.Figure> list = new SimpleListStack.SimpleList<Figures.Figure>();

list.Add(circle);

list.Add(rect);

list.Add(square);

Console.WriteLine("\nПеред сортировкой:");

foreach (var x in list) Console.WriteLine(x);

list.Sort();

Console.WriteLine("\nПосле сортировки:");

foreach (var x in list) Console.WriteLine(x);

Console.WriteLine("\nСтек");

SimpleListStack.SimpleStack<Figures.Figure> stack = new SimpleListStack.SimpleStack<Figures.Figure>();

//добавление данных в стек

stack.Push(rect);

stack.Push(square);

stack.Push(circle);

//чтение данных из стека

while (stack.Count > 0)

{

Figures.Figure f = stack.Pop();

Console.WriteLine(f);

}

Console.ReadLine();

}

}

}

**4). Пример выполнения**

