**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3

«Трек курса «Задания на основе языка C#»»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-34Б |  | преподаватель каф. ИУ5 |
| Киреев А.А. |  | Ю.Е. Гапанюк. |
| Подпись и дата:  28.12.2020 |  | Подпись и дата:  28.12.2020 |

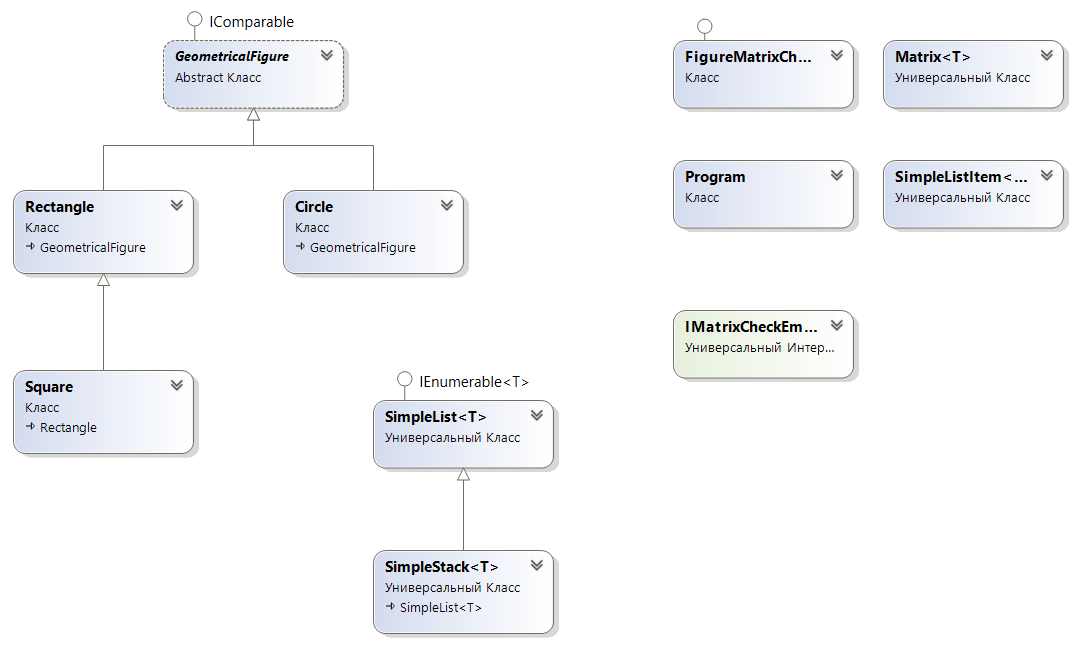
Москва, 2020 г.

**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**

using System;

using System.Collections.Generic;

using System.Collections;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Kireev\_IU5\_34\_Lab3

{

abstract class GeometricalFigure : IComparable //Абстрактный класс

{

public virtual double Area()

{

return 0;

}

public abstract void Print();

public int CompareTo(object obj)

{

GeometricalFigure p = (GeometricalFigure)obj;

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

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

else return 1;

}

public string Type

{

get

{

return this.\_Type;

}

protected set

{

this.\_Type = value;

}

}

string \_Type;

public override string ToString()

{

return this.Type + " площадью " + this.Area().ToString();

}

}

class Rectangle : GeometricalFigure //Прямоугольник

{

private double \_length;

private double \_width;

public double length

{

get { return \_length; }

set { \_length = value; }

}

public double width

{

get { return \_width; }

set { \_width = value; }

}

public Rectangle(double l, double w)

{

this.length = l;

this.width = w;

}

public override double Area() //Находим площадь

{

return \_length \* \_width;

}

public override string ToString() //Переопределяем Object.ToString()

{

return "Фигура: Прямоугольник\n" + "Длина = " + length.ToString() + "\n" + "Ширина = " + width.ToString() + "\n" + "Площадь = " + this.Area() + "\n";

}

public override void Print()

{

Console.WriteLine(this.ToString());

}

}

class Square : Rectangle //Квадрат

{

public Square(double l)

: base(l, l)

{

}

public override string ToString()

{

return "Фигура: Квадрат\n" + "Длина стороны = " + length.ToString() + "\n" + "Площадь = " + this.Area() + "\n";

}

public override void Print()

{

Console.WriteLine(this.ToString());

}

}

class Circle : GeometricalFigure //Круг

{

private double \_radius;

public double radius

{

get

{

return \_radius;

}

set

{

\_radius = value;

}

}

public Circle(double r)

{

this.radius = r;

}

public override double Area()

{

return Math.PI \* \_radius \* \_radius;

}

public override string ToString()

{

return "Фигура: Круг\n" + "Радиус = " + radius.ToString() + "\n" + "Площадь = " + this.Area() + "\n";

}

public override void Print()

{

Console.WriteLine(this.ToString());

}

}

public class Matrix<T> {

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

int maxX;

int maxY;

int maxZ;

//Bнтерфейс для проверки пустого элемента

IMatrixCheckEmpty<T> сheckEmpty;

// Конструктор

public Matrix(int px, int py, int pz, IMatrixCheckEmpty<T> сheckEmptyParam)

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.сheckEmpty = сheckEmptyParam;

}

// Индексатор для доступа к данных

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.сheckEmpty.getEmptyElement();

}

}

}

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.maxX)

{

throw new ArgumentOutOfRangeException("z", "z=" + z + " выходит за границы");

}

}

// Формирование ключа

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

{

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

}

// Приведение к строке

public override string ToString()

{

StringBuilder b = new StringBuilder();

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

{

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\t");

}

//Если текущий элемент не пустой

if (!this.сheckEmpty.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("]\n");

}

}

return b.ToString();

}

}

// Проверка пустого элемента матрицы

public interface IMatrixCheckEmpty<T>

{

// Возвращает пустой элемент

T getEmptyElement();

// Проверка что элемент является пустым

bool checkEmptyElement(T element);

}

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<GeometricalFigure>

{

// В качестве пустого элемента возвращается null

public GeometricalFigure getEmptyElement()

{

return null;

}

// Проверка что переданный параметр равен null

public bool checkEmptyElement(GeometricalFigure element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

}

public class SimpleListItem<T>

{

// Данные

public T data { get; set; }

// Следующий элемент

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

//конструктор

public SimpleListItem(T param)

{

this.data = param;

}

}

public class SimpleList<T> : IEnumerable<T> where T : IComparable

{

// Первый элемент списка

protected SimpleListItem<T> first = null;

// Последний элемент списка

protected SimpleListItem<T> last = null;

// Количество элементов

public int Count

{

get

{

return \_count;

}

protected set

{

\_count = value;

}

}

int \_count;

// Добавление элемента

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;

}

}

// Чтение контейнера с заданным номером

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;

}

// Чтение элемента с заданным номером

public T Get(int number)

{

return GetItem(number).data;

}

// Для перебора коллекции

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();

}

// Cортировка

public void Sort()

{

Sort(0, this.Count - 1);

}

// Алгоритм быстрой сортировки

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);

}

// Вспомогательный метод для обмена элементов при сортировке

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;

}

}

// Класс стек

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

{

// Добавление в стек

public void Push(T element)

{

Add(element);

}

// Удаление и чтение из стека

public T Pop()

{

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;

}

}

class Program

{

static void Main(string[] args)

{

Rectangle R1 = new Rectangle(5, 9);

Square S1 = new Square(6);

Circle C1 = new Circle(8);

ArrayList list = new ArrayList();

list.Add(R1);

list.Add(S1);

list.Add(C1);

Console.WriteLine("--------------------------------------");

Console.WriteLine("ArrayList:");

Console.WriteLine("Перед сортировкой:");

foreach (GeometricalFigure figure in list)

{

figure.Print();

}

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

list.Sort();

Console.WriteLine("После сортировки: ");

foreach (GeometricalFigure figure in list)

{

figure.Print();

}

Console.WriteLine("--------------------------------------");

List<GeometricalFigure> figureList = new List<GeometricalFigure>();

figureList.Add(R1);

figureList.Add(S1);

figureList.Add(C1);

Console.WriteLine("List<T>:");

Console.WriteLine("Перед сортировкой:");

foreach (GeometricalFigure figure in figureList)

{

figure.Print();

}

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

figureList.Sort();

Console.WriteLine("После сортировки: ");

foreach (GeometricalFigure figure in figureList)

{

figure.Print();

}

Console.WriteLine("--------------------------------------");

Console.WriteLine("Матрица");

Matrix<GeometricalFigure> matrix = new Matrix<GeometricalFigure>(3, 3, 3, new FigureMatrixCheckEmpty());

matrix[0, 0, 0] = R1;

matrix[1, 1, 1] = S1;

matrix[2, 2, 2] = C1;

Console.WriteLine(matrix.ToString());

Console.WriteLine("--------------------------------------");

SimpleStack<GeometricalFigure> stack = new SimpleStack<GeometricalFigure>();

//добавление данных в стек

stack.Push(R1);

stack.Push(S1);

stack.Push(C1);

//чтение данных из стека

while (stack.Count > 0)

{

GeometricalFigure figure = stack.Pop();

Console.WriteLine(figure);

}

Console.WriteLine("--------------------------------------");

Console.Read();

}

}

}

**Экранные формы с примерами выполнения программы**

