**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

«Реализация работы c коллекциями»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Олейников И.И. |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

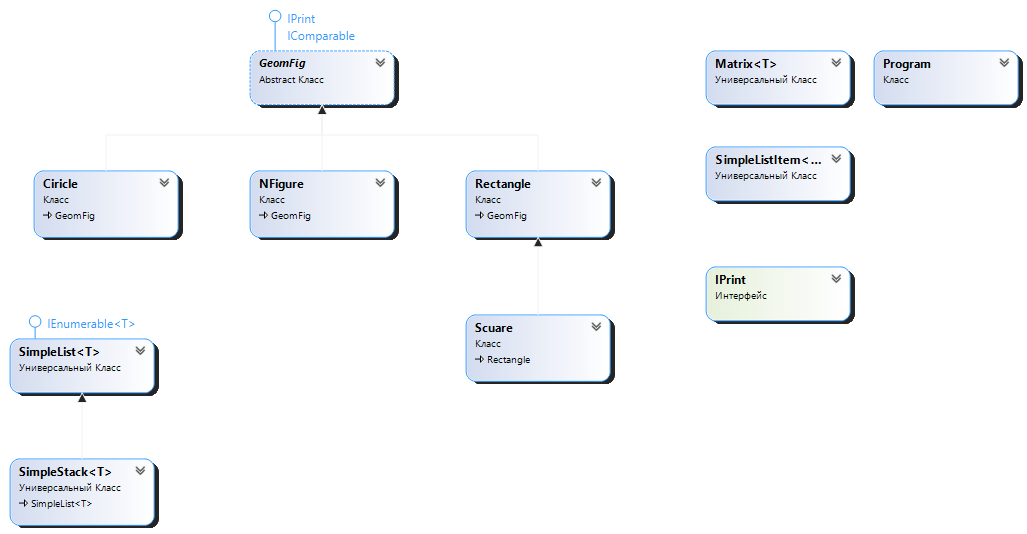
Москва, 2020 г.

**Условие 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:**

**Class1**

using System;

using System.Collections.Generic;

using System.Text;

namespace \_3

{

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;

}

}

public class SimpleStack<T> : SimpleList<T>

where T : IComparable

{

public void Push(T element)

{

Add(element);

}

public SimpleListItem<T> Pop()

{

SimpleListItem<T> item;

if (this.Count == 0)

throw new Exception("В стеке больше нет элементов");

if (this.Count == 1)

{

item = this.first;

this.first = null;

this.last = null;

this.Count = 0;

return item;

}

//если элементов >1

item = this.last;

//this.last = null;

this.last = this.GetItem(this.Count - 2);

this.Count--;

return item;

}

}

}

**Class2**

using System;

using System.Collections.Generic;

using System.Text;

namespace \_3

{

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>

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

/// </summary>

int maxZ;

/// <summary>

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

/// </summary>

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);

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

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

{

return this.\_matrix[key];

}

else

{

return this.nullElement;

}

}

set

{

CheckBounds(x, y, z);

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

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

}

}

/// <summary>

/// Проверка границ

/// </summary>

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

{

if (x < 0 || x >= this.maxX) throw new Exception("x=" + x + " выходит за границы");

if (y < 0 || y >= this.maxY) throw new Exception("y=" + y + " выходит за границы");

if (z < 0 || z >= this.maxZ) throw new Exception("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()

{

///Класс StringBuilder используется для построения длинных строк

///Это увеличивает производительность по сравнению с созданием и склеиванием

///большого количества обычных строк

StringBuilder b = new StringBuilder();

for (int k = 0; k != maxZ; ++k)

{

b.Append($"z={k}\n");

for (int j = 0; j != maxY; ++j)

{

b.Append("[ ");

for (int i = 0; i != maxX; ++i)

{

if (i > 0) b.Append(" | ");

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

}

b.Append(" ]\n");

}

b.Append("\n\n\n\n");

}

return b.ToString();

}

}

}

**Class3**

using System;

using System.Collections.Generic;

//using System.Text;

namespace \_3

{

#region Абстрактные

interface IPrint

{

void Print();

}

abstract class GeomFig : IPrint, IComparable

{

public int CompareTo(GeomFig other)

{

if (other == null)

throw new Exception("Ошибка! Невозможно сравнить два объекта");// ручная генерация исключения

return S.CompareTo(other.S);

}

public int CompareTo(object other)

{

if (other == null)

throw new Exception("Ошибка! Невозможно сравнить два объекта");// ручная генерация исключения

return CompareTo(other as GeomFig);

}

public void Print()//вывод

{

Console.WriteLine(Out());

}

public virtual void Sum()// нахождение площади

{

}

public virtual string Out()//вывод

{

return "Площадь фигуры равна:" + S.ToString();

}

public double S { get; set; }//свойство

public string Name { get; set; }//свойство

}

#endregion

class NFigure : GeomFig

{

public override string ToString()

{

return "\tNone\t";

}

}

#region Прямоугольник

class Rectangle : GeomFig//прямоугольник

{

public Rectangle() { }

public Rectangle(double h, double w)//конструктор

{

Height = h;

Width = w;

Name = "Прямоугольник";

}

public double Height { get; set; }//свойство

public double Width { get; set; }//свойство

public override void Sum()

{

S = Height \* Width;

}

public override string Out()//переопр вирт метода возвр значений

{

return "Прямоугольник- Длина равна: " + Height.ToString() + " Ширина равна: " + Width.ToString() + " " + base.Out();

}

}

#endregion

#region Квадрат

class Scuare : Rectangle //квадрат

{

public Scuare(double h)//конструктор

{

Height = h;

Name = "Квадрат";

}

public override void Sum()

{

S = Height \* Height;

}

public override string Out()//переопр вирт метода возвр значений

{

return "Квадрат- Длина равна: " + Height.ToString() + " Площадь фигуры равна:" + S.ToString();

}

}

#endregion

#region Круг

class Ciricle : GeomFig//круг

{

public Ciricle(double c)//конструктор

{

Rad = c;

Name = "Круг";

}

public double Rad { get; set; }//свойство

public override void Sum()//мереопр вирт метода подсчета площади

{

S = Math.PI \* Rad \* Rad;

}

public override string Out()//переопр вирт метода возвр значений

{

return "Круг- Радиус равен: " + Rad.ToString() + " " + base.Out();

}

}

#endregion

}

**Programm**

using System;

using System.Collections;

using System.Collections.Generic;

namespace \_3

{

class Program

{

static void Main(string[] args)

{

double t1, t2;

#region Ввод даных прямоуольника

Console.WriteLine("Введите первую сторону прямоугольника");

while (!double.TryParse(Console.ReadLine(), out t1) || t1 < 0)//если ввод- число, записываем

Console.WriteLine("Ввод неверен, попробуйте еще раз");

Console.WriteLine("Введите вторую сторону прямоугольника");

while (!double.TryParse(Console.ReadLine(), out t2) || t2 < 0)//если ввод- число, записываем

Console.WriteLine("Ввод неверен, попробуйте еще раз");

#endregion

Rectangle rec = new Rectangle(t1, t2);// создание объекта класса прямоугольник

#region Ввод данных квадрата

Console.WriteLine("Введите сторону квадрата");

while (!double.TryParse(Console.ReadLine(), out t1) || t1 < 0)//если ввод- число, записываем

Console.WriteLine("Ввод неверен, попробуйте еще раз");

#endregion

Scuare scuar = new Scuare(t1);// создание объекта класса квадрат

#region Ввод данных круга

Console.WriteLine("Введите радиус круга");

while (!double.TryParse(Console.ReadLine(), out t1) || t1 < 0)//если ввод- число, записываем

Console.WriteLine("Ввод неверен, попробуйте еще раз");

#endregion

Ciricle cir = new Ciricle(t1);// создание объекта класса круг

rec.Sum();// подсчет площади прямоугольника

scuar.Sum();// подсчет площади квадрата

cir.Sum();// подсчет площади круга

#region Работа с ArrayList

ArrayList arlist = new ArrayList() { rec, scuar, cir };//создаем ArrayList и добавляем в него объекты

arlist.Sort();// сортируем по площади с пом. CompereTo

Console.WriteLine();//вывод инфы

Console.WriteLine("Сортировка ArrayList по возрастанию");

foreach (object other in arlist)

Console.WriteLine($"{other}");

#endregion

#region Работа с List

List<GeomFig> list = new List<GeomFig>() { rec, scuar, cir};// создаем list и добавляем объекты

list.Sort();

Console.WriteLine();

Console.WriteLine("Сортировка List по возрастанию");

for (int i = 0; i < list.Count; i++)

{

Console.WriteLine($"{i+1}-{list[i].Name} с площадью: {list[i].S}"); ;

}

#endregion

//SimpleStack sim= new SimpleStack();

//sim.Push(cir);

Console.WriteLine();

rec.Print();//вывод инормации

scuar.Print();//вывод инормации

cir.Print();//вывод инормации

#region Вывод матрицы

Console.WriteLine("\nРабота с матрицей\n");

Matrix<GeomFig> FigMatrix = new Matrix<GeomFig>(6, 5, 6, new NFigure());

FigMatrix[0, 0, 1] = rec;

FigMatrix[2, 1, 0] = scuar;

FigMatrix[3, 0, 4] = cir;

FigMatrix[1, 2, 0] = rec;

FigMatrix[4, 4, 1] = cir;

FigMatrix[5, 1, 3] = scuar;

Console.WriteLine("\n" + FigMatrix);

#endregion

#region Стек

Console.WriteLine("\nРабота со стеком:\n");

SimpleStack<GeomFig> GeomStack = new SimpleStack<GeomFig>();

GeomStack.Push(rec);//вносим наши фигуры

GeomStack.Push(cir);

GeomStack.Push(scuar);

GeomStack.Push(scuar);

GeomStack.Push(cir);

while (GeomStack.Count != 0)//выводим их

GeomStack.Pop().data.Print();

#endregion

Console.ReadLine();

}

}

}

**Примеры выполнения программы:**

