|  |  |
| --- | --- |
|  | **Министерство образования и науки 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: **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА: **СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ**

**ОТЧЁТ ПО Лабораторной работе №3 по курсу**

**“Базовые компоненты интернет-технологий”**

Студентка Алиева Довлат Гусейн кызы

Группа ИУ5-31

*2017 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.

Пример работы класса SimpleStack реализовать на основе геометрических фигур.

**Текст программы**

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Collections;

namespace Figures

{

class Program

{

static void Main(string[] args)

{

Rect rect = new Rect(6, 4);

Square square = new Square(5);

Circle circle = new Circle(7);

// ArrayList

Console.WriteLine("\nArrayList");

ArrayList arrayList = new ArrayList();

arrayList.Add(circle);

arrayList.Add(rect);

arrayList.Add(square);

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

Console.WriteLine("\nArrayList - отсортированный");

arrayList.Sort();

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

// List<Figure>

Console.WriteLine("\nList<AbstractFigure>");

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

figureList.Add(circle);

figureList.Add(rect);

figureList.Add(square);

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

Console.WriteLine("\nList<AbstractFigure> - отсортированный");

arrayList.Sort();

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

// Matrix<Figure>

Console.WriteLine("\nMatrix<AbstractFigure>");

Matrix<AbstractFigure> matrix = new Matrix<AbstractFigure>(2, 2, 2, rect);

Console.WriteLine(matrix.ToString());

// SimpleList<AbstractFigure>

Console.WriteLine("SimpleList<AbstractFigure>");

SimpleList<AbstractFigure> list = new SimpleList<AbstractFigure>();

list.Add(square);

list.Add(rect);

list.Add(circle);

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

Console.WriteLine("\nSimpleList<AbstractFigure> - отсортированный");

list.Sort();

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

// SimpleStack<AbstractFigure>

Console.WriteLine("\nSimpleStack<AbstractFigure>");

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

stack.Push(rect);

stack.Push(square);

stack.Push(circle);

while (stack.Count > 0)

{

AbstractFigure f = stack.Pop();

Console.WriteLine(f);

}

Console.ReadKey();

}

}

}

**AbstractFigure.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Figures

{

/// <summary>

/// Класс фигура

/// </summary>

abstract class AbstractFigure : IComparable

{

/// <summary>

/// Тип фигуры

/// </summary>

public string Type

{

get

{

return this.\_Type;

}

protected set

{

this.\_Type = value;

}

}

string \_Type;

/// <summary>

/// Вычисление площади

/// </summary>

/// <returns></returns>

public abstract double Area();

/// <summary>

/// Приведение к строке, переопределение метода Object

/// </summary>

/// <returns></returns>

public override string ToString()

{

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

}

/// <summary>

/// Сравнение элементов (для сортировки списка)

/// </summary>

/// <param name="obj"></param>

/// <returns></returns>

public int CompareTo(object obj)

{

AbstractFigure p = (AbstractFigure)obj;

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

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

else return 1;

}

}

}

**Square.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Figures

{

class Square : Rect, IPrint

{

public Square(double size)

: base(size, size)

{

this.Type = "Квадрат";

}

}

}

**Circle.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Figures

{

class Circle : AbstractFigure, IPrint

{

/// <summary>

/// Ширина

/// </summary>

double radius;

/// <summary>

/// Основной конструктор

/// </summary>

/// <param name="pr">Радиус</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());

}

}

}

**Rect.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Figures

{

class Rect : AbstractFigure, IPrint

{

/// <summary>

/// Высота

/// </summary>

double height;

/// <summary>

/// Ширина

/// </summary>

double width;

/// <summary>

/// Основной конструктор

/// </summary>

/// <param name="height">Высота</param>

/// <param name="width">Ширина</param>

public Rect(double height, double width)

{

this.height = height;

this.width = width;

this.Type = "Прямоугольник";

}

/// <summary>

/// Вычисление площади

/// </summary>

public override double Area()

{

return this.width \* this.height;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

**IPrintInterface.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Figures

{

interface IPrint

{

void Print();

}

}

**Matrix.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Figures

{

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 в глубину

/// </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)

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.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 < this.maxY; k++)

{

b.Append("[");

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

{

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

b.Append("[");

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

{

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

if (i != (this.maxX - 1)) b.Append(", ");

}

b.Append("]");

}

b.Append("]\n");

}

return b.ToString();

}

}

}

**SimpleList.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Figures

{

/// <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>

/// Реализация алгоритма 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.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Figures

{

class SimpleStack<T> : SimpleList<T>

where T : IComparable

{

/// <summary>

/// Добавление в стек

/// </summary>

public void Push(T element)

{

Add(element);

}

/// <summary>

/// Чтение с удалением из стека

/// </summary>

public T Pop()

{

T element = Get(Count-1);

SimpleListItem<T> listItem = GetItem(Count - 1);

listItem = null;

Count--;

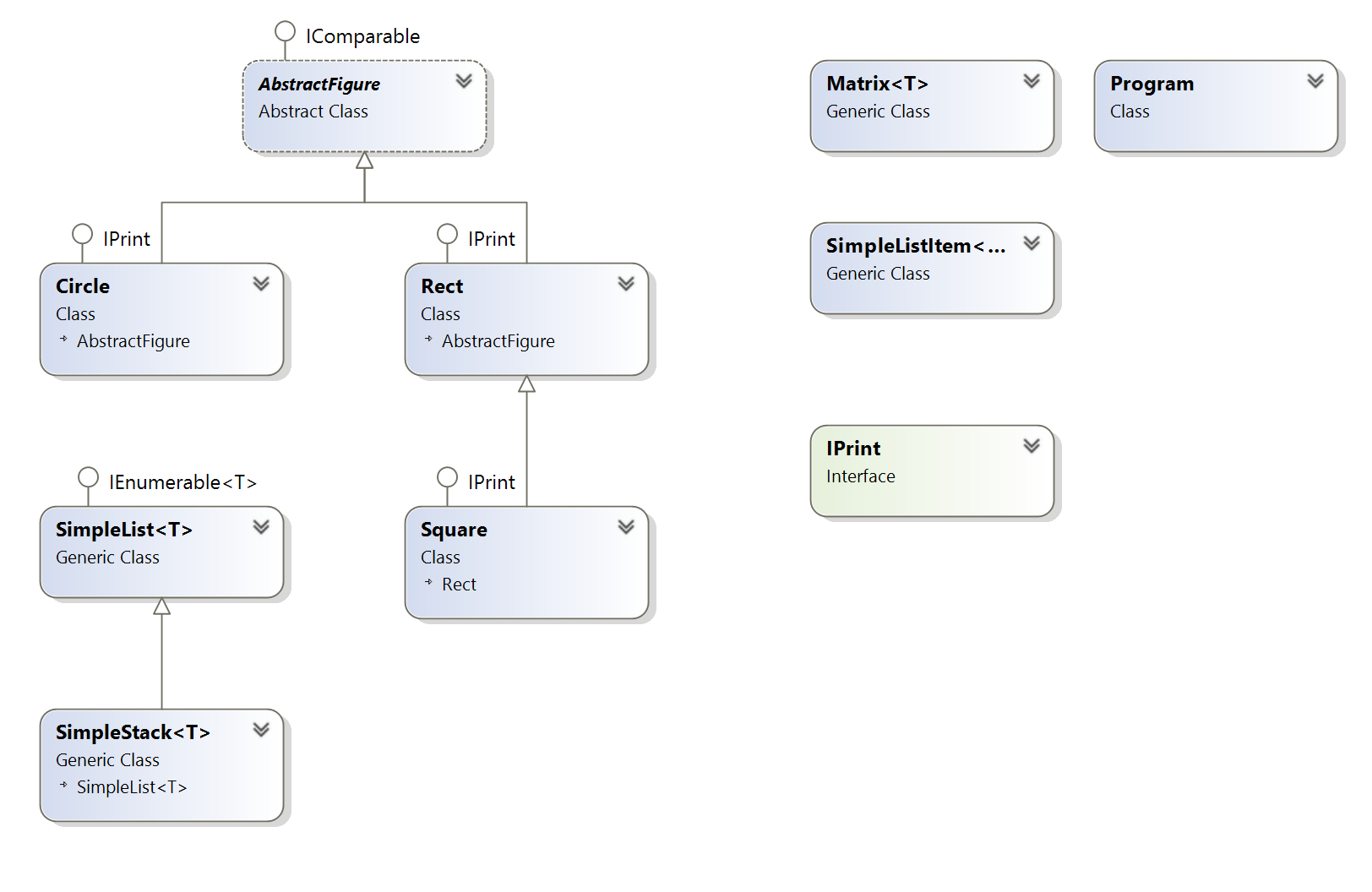
return element;

}

}

}

**Диаграмма классов**



**Результаты выполнения**

