|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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)** |

Базовые компоненты интернет технологий

Отчет по лабораторной работе №3

Студент: Пылаев Б. А.

Группа: ИУ5Ц-52Б

Преподаватель: Гапанюк Ю. Е.

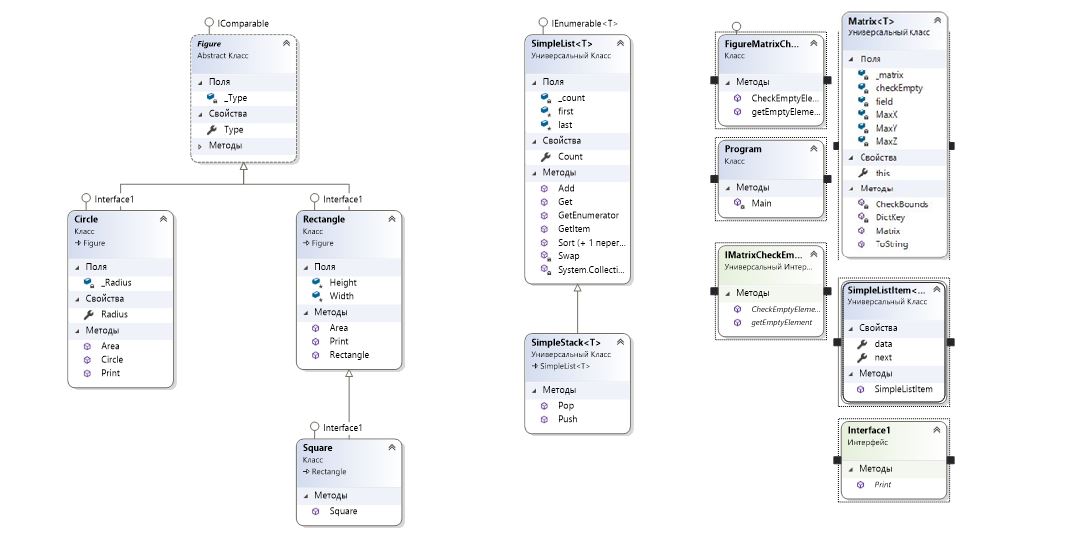
2020 г.

**Лабораторная работа №3**

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**

Program.cs

using FigureCollections;

using System;

using System.Collections;

using System.Collections.Generic;

using System.Net.Http.Headers;

namespace LAB\_3

{

class Program

{

static void printArrayList(ArrayList array)

{

foreach (object obj in array)

{

string figureType = obj.GetType().Name;

if (figureType == "Rectangle")

{

Rectangle rect = obj as Rectangle;

rect.Print();

}

else if (figureType == "Square")

{

Square square = obj as Square;

square.Print();

}

else if (figureType == "Circle")

{

Circle circle = obj as Circle;

circle.Print();

}

}

Console.WriteLine();

}

static void printList(List<Figure> list)

{

foreach (object obj in list)

{

string figureType = obj.GetType().Name;

if (figureType == "Rectangle")

{

Rectangle rect = obj as Rectangle;

rect.Print();

}

else if (figureType == "Square")

{

Square square = obj as Square;

square.Print();

}

else if (figureType == "Circle")

{

Circle circle = obj as Circle;

circle.Print();

}

}

Console.WriteLine();

}

static void Main(string[] args)

{

Rectangle rect = new Rectangle(3, 2);

Square square = new Square(4);

Circle circle = new Circle(3);

ArrayList array = new ArrayList();

array.Add(rect);

array.Add(square);

array.Add(circle);

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("ArrayList. До сортировки: ");

Console.ResetColor();

printArrayList(array);

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("ArrayList. После сортировки: ");

Console.ResetColor();

array.Sort();

printArrayList(array);

//

List<Figure> figuresList = new List<Figure>

{

rect,

square,

circle

};

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("List<Figure>. До сортировки: ");

Console.ResetColor();

printList(figuresList);

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("List<Figure>. После сортировки: ");

Console.ResetColor();

figuresList.Sort();

printList(figuresList);

//

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Разреженная матрица. ");

Console.ResetColor();

SparseMatrix<Figure> matrix = new SparseMatrix<Figure>(5, 5, 5, rect);

matrix[0, 0, 0] = square;

matrix[2, 2, 2] = circle;

try

{

matrix[100, 100, 100] = rect;

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

Console.WriteLine(matrix);

//

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("SimpleStack для фигур. ");

Console.ResetColor();

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

stack.Push(rect);

stack.Push(square);

stack.Push(circle);

while (stack.Count > 0)

{

Figure figure = stack.Pop();

Console.WriteLine(figure);

}

//Console.WriteLine("Квадрат и круг: ", square.CompareTo(circle).ToString());

}

}

}

Figures.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace LAB\_3

{

interface IPrint

{

void Print()

{

}

}

abstract class Figure : IComparable

{

public double area;

public virtual double calcArea(double height, double width)

{

return area = height \* width;

}

public virtual double calcArea(double radius)

{

return area = Math.PI \* (radius \* radius);

}

public int CompareTo(object obj)

{

Figure figure = obj as Figure;

if (this.area > figure.area)

return 1;

else if (this.area < figure.area)

return -1;

else

return 0;

}

}

class Rectangle : Figure, IPrint

{

public double height

{

get;

set;

}

public double width

{

get;

set;

}

public Rectangle(double Height, double Width)

{

height = Height;

width = Width;

calcArea(height, width);

}

public override string ToString()

{

return String.Concat("[Прямоугольник]. Высота: ", height, ", ширина: ", width, ", площадь: ", area);

}

public void Print()

{

Console.WriteLine(ToString());

}

}

class Square : Rectangle, IPrint

{

public double side

{

get;

set;

}

public Square(double Side) : base(Side, Side)

{

side = Side;

calcArea(side, side);

}

public override string ToString()

{

return String.Concat("[Квадрат]. Сторона: ", side, ", площадь: ", area);

}

public new void Print()

{

Console.WriteLine(ToString());

}

}

class Circle : Figure, IPrint

{

public double radius

{

get;

set;

}

public Circle(double Radius)

{

radius = Radius;

calcArea(radius);

}

public override string ToString()

{

return String.Concat("[Окружность]. Радиус: ", radius, ", площадь: ", area);

}

public void Print()

{

Console.WriteLine(ToString());

}

}

}

SimpleList.cs

using System;

using System.Collections.Generic;

namespace FigureCollections

{

/// <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>

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 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 FigureCollections

{

/// <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;

namespace FigureCollections

{

/// <summary>

/// Класс стек

/// </summary>

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

{

/// <summary>

/// Добавление в стек

/// </summary>

public void Push(T element)

{

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

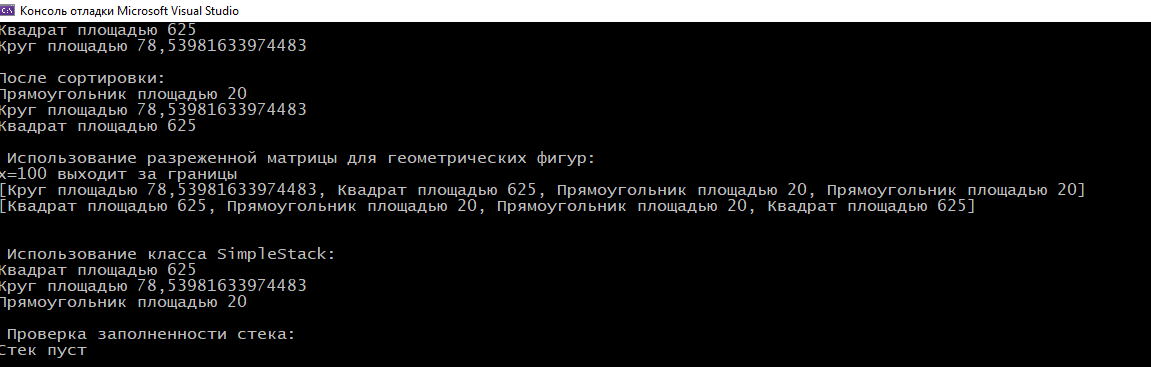
return Result;

}

}

}

**Тест программы**



**Cсылка на репозиторий исходных кодов GitHub**

https://github.com/bogdanpylaev/BCIT\_Bogdan\_Pylaev