|  |  |
| --- | --- |
| 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: Карпов И.Е.

Группа: ИУ5Ц-54Б

Преподаватель: Гапанюк Ю. Е.

2020 г.

**Лабораторная работа №3**

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.

**Текст программы**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**using System.Threading.Tasks;**

**using System.Collections;**

**namespace ConsoleApplication4**

**{**

**abstract partial class Geometric\_figure : IComparable**

**{**

**private string \_Type;**

**/// <summary>**

**/// Название фигуры**

**/// </summary>**

**public string Type**

**{**

**get { return this.\_Type; }**

**set { this.\_Type = value; }**

**}**

**/// <summary>**

**/// Вычисление площади**

**/// </summary>**

**/// <returns></returns>**

**abstract public double Area();**

**public override string ToString()**

**{**

**//Console.WriteLine(this.Type + ":");**

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

**}**

**public int CompareTo(object obj)**

**{**

**Geometric\_figure p = (Geometric\_figure)obj;**

**if (this.Area() > p.Area())**

**{**

**return 1;**

**}**

**else if (this.Area() < p.Area())**

**{**

**return -1;**

**}**

**else if (this.Area() == p.Area())**

**{**

**return 0;**

**}**

**else**

**{**

**throw new NotImplementedException();**

**}**

**}**

**}**

**/// Класс Квадрат**

**interface Interface1**

**{**

**void Print();**

**}**

**class Square : Geometric\_figure, Interface1**

**{**

**/// Радиус**

**//protected double Radius;**

**private double \_Storon;**

**public double Storon**

**{**

**get { return \_Storon; }**

**set { this.\_Storon = value; }**

**}**

**/// Конструктор**

**public Square(double storon)**

**{**

**this.Storon = storon;**

**this.Type = "Квадрат";**

**}**

**/// Площадь Круга**

**public override double Area()**

**{**

**return this.Storon \* this.Storon;**

**}**

**///// Переопределенный метод преобразования в строку**

**//public override string ToString()**

**//{**

**// return "Круг: радиус = " + this.Radius + "; площадь = " + this.Area();**

**//}**

**/// Метод вывода на консоль**

**public void Print()**

**{**

**Console.WriteLine(this.ToString());**

**Console.WriteLine("Сторона: " + this.Storon);**

**}**

**}**

**/// Класс Прямоугольник**

**class Rectangle : Geometric\_figure, Interface1**

**{**

**/// Ширина прямоугольника**

**protected double Width;**

**/// Высота прямоугольника**

**protected double Height;**

**/// Конструктор**

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

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

**public Rectangle(double width, double height)**

**{**

**this.Width = width;**

**this.Height = height;**

**this.Type = "Прямоугольник";**

**}**

**/// Площадь прямоугольника**

**public override double Area()**

**{**

**return Width \* Height;**

**}**

**///// Переопределенный метод преобразования в строку**

**//public override string ToString()**

**//{**

**// return "Прямоугольник: ширина = " + this.Width + "; высота = " + this.Height +**

**// "; площадь: " + this.Area();**

**//}**

**/// Метод вывода на консоль**

**public void Print()**

**{**

**Console.WriteLine(this.ToString());**

**Console.WriteLine("Высота: " + this.Height);**

**Console.WriteLine("Ширина: " + this.Width);**

**}**

**}**

**/// Класс Круг**

**class Circle : Geometric\_figure, Interface1**

**{**

**/// Радиус**

**//protected double Radius;**

**private double \_Radius;**

**public double Radius**

**{**

**get { return \_Radius; }**

**set { this.\_Radius = value; }**

**}**

**/// Конструктор**

**public Circle(double radius)**

**{**

**this.Radius = radius;**

**this.Type = "Круг";**

**}**

**/// Площадь Круга**

**public override double Area()**

**{**

**return this.Radius \* this.Radius \* Math.PI;**

**}**

**///// Переопределенный метод преобразования в строку**

**//public override string ToString()**

**//{**

**// return "Круг: радиус = " + this.Radius + "; площадь = " + this.Area();**

**//}**

**/// Метод вывода на консоль**

**public void Print()**

**{**

**Console.WriteLine(this.ToString());**

**Console.WriteLine("Радиус: " + this.Radius);**

**}**

**partial class Matrix<T>**

**{**

**///<summary>**

**/// Словарь для хранения значений**

**/// </summary>**

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

**///<summary>**

**///Макс. количество столбцов**

**/// </summary>**

**int MaxX;**

**///<summary>**

**///Макс.количество строк**

**///</summary>**

**int MaxY;**

**///<summary>**

**///Макс. количество столбцов**

**/// </summary>**

**int MaxZ;**

**///<summary>**

**///Реализация интерфейса для проверки пустого элемента**

**///</summary>**

**IMatrixCheckEmpty<T> checkEmpty;**

**///<summary>**

**///Конструктор**

**/// </summary>**

**public Matrix(int x, int y, int z, IMatrixCheckEmpty<T> param)**

**{**

**this.MaxX = x;**

**this.MaxY = y;**

**this.MaxZ = z;**

**this.checkEmpty = param;**

**}**

**///<summary>**

**///Индексатор для доступа к данным**

**/// </summary>**

**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.checkEmpty.getEmptyElement();**

**}**

**}**

**}**

**///<summary>**

**///Проверка границ**

**///</summary>**

**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.MaxZ)**

**{**

**throw new ArgumentOutOfRangeException("z", "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>**

**///Преобразование ToString() по 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++)**

**for (int o = 0; o < this.MaxZ; o++)**

**{**

**if (i > 0)**

**{**

**b.Append("\t");**

**}**

**//если элемент не пустой**

**if (!this.checkEmpty.CheckEmptyElement(this[i, j, o]))**

**{**

**//добавить этот элемент, преобразованный в строку**

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

**}**

**//иначе добавить "пусто"**

**else**

**{**

**b.Append(" - ");**

**}**

**}**

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

**}**

**return b.ToString();**

**}**

**}**

**///<summary>**

**///Список**

**/// </summary>**

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

**{**

**///<summary>**

**///Первый элемент списка**

**/// </summary>**

**protected SimpleListItem<T> first = null;**

**///<summary>**

**///Послдений 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;**

**//присоединенный 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;**

**}**

**}**

**/// <summary>**

**/// класс стек**

**/// </summary>**

**partial 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 - значение по умолчанию**

**T Result = default(T);**

**if (this.Count == 0)**

**{**

**return Result;**

**}**

**if (this.Count == 1)**

**{**

**Result = this.first.data;**

**this.first = null;**

**this.last = null;**

**}**

**else**

**{**

**//поиск предпоследнего элемента**

**SimpleListItem<T> newLast = this.GetItem(this.Count - 2);**

**//чтение из последнего элемента**

**Result = newLast.next.data;**

**//предпоследний элемент считается последним**

**this.last = newLast;**

**//последний элемент удаляется**

**newLast.next = null;**

**}**

**//уменьшение количества элементов в списке**

**this.Count--;**

**//возврат результата**

**return Result;**

**}**

**}**

**/// <summary>**

**/// Элемент списка**

**/// </summary>**

**partial 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;**

**}**

**}**

**partial class Geometric\_figureMatrixCheckEmpty : IMatrixCheckEmpty<Geometric\_figure>**

**{**

**//реализация первого метода интерфейса**

**public Geometric\_figure getEmptyElement()**

**{**

**return null;**

**}**

**public bool CheckEmptyElement(Geometric\_figure element)**

**{**

**bool Result = false;**

**if (element == null)**

**{**

**Result = true;**

**}**

**return Result;**

**}**

**}**

**//методы данного интерфейса используются при создании разреженной матрицы**

**public interface IMatrixCheckEmpty<T>**

**{**

**//возвращает пустой элемент**

**T getEmptyElement();**

**//проверка, что элемент является пустым**

**bool CheckEmptyElement(T element);**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**Console.WriteLine("Лабораторная работа №3");**

**// Объект класса Rectangle**

**Rectangle rect = new Rectangle(2, 4);**

**rect.Print();**

**// Объект класса**

**Square square = new Square(5);**

**square.Print();**

**// Объект класса**

**Circle circle = new Circle(3);**

**circle.Print();**

**//коллекция класса ArrayList**

**ArrayList Geometric\_figures = new ArrayList();**

**Geometric\_figures.Add(circle);**

**Geometric\_figures.Add(rect);**

**Geometric\_figures.Add(square);**

**Console.WriteLine("\nДо сортировки для ArrayList");**

**foreach (var i in Geometric\_figures)**

**{**

**Console.WriteLine(i);**

**}**

**Geometric\_figures.Sort();**

**Console.WriteLine("\nПосле сортировки для ArrayList");**

**foreach (var i in Geometric\_figures)**

**{**

**Console.WriteLine(i);**

**}**

**//коллекция класса List<Geometric\_figure>**

**List<Geometric\_figure> Geometric\_figures1 = new List<Geometric\_figure>();**

**Geometric\_figures1.Add(circle); //добавление в коллекцию**

**Geometric\_figures1.Add(rect);**

**Geometric\_figures1.Add(square);**

**Console.WriteLine("\n\nДо сортировки для List<Geometric\_figure>:");**

**foreach (var i in Geometric\_figures1)**

**{**

**Console.WriteLine(i);**

**}**

**Console.WriteLine("\nПосле сортировки для List<Geometric\_figure>:");**

**Geometric\_figures1.Sort();**

**foreach (var i in Geometric\_figures1)**

**{**

**Console.WriteLine(i);**

**}**

**//создание разреженной матрицы**

**//создание разреженной матрицы**

**Console.WriteLine("\n\nМатрица:");**

**Matrix<Geometric\_figure> matrix = new Matrix<Geometric\_figure>(3, 3, 3, new Geometric\_figureMatrixCheckEmpty());**

**matrix[0, 0, 0] = rect;**

**matrix[1, 1, 1] = square;**

**matrix[2, 2, 2] = circle;**

**Console.WriteLine(matrix.ToString());**

**//использование коллекции SimpleList**

**SimpleList<Geometric\_figure> list = new SimpleList<Geometric\_figure>();**

**list.Add(circle);**

**list.Add(rect);**

**list.Add(square);**

**Console.WriteLine("\n\nПеред сортировкой (SimpleList):");**

**foreach (var a in list)**

**{**

**Console.WriteLine(a);**

**}**

**list.Sort();**

**Console.WriteLine("\n\nПосле сортировки (SimpleList):");**

**foreach (var a in list)**

**{**

**Console.WriteLine(a);**

**}**

**//использование собственного стека**

**SimpleStack<Geometric\_figure> stack = new SimpleStack<Geometric\_figure>();**

**stack.Push(circle);**

**stack.Push(rect);**

**stack.Push(square);**

**Console.WriteLine("\n\nИспользование стека:");**

**while (stack.Count > 0)**

**{**

**Geometric\_figure f = stack.Pop();**

**Console.WriteLine(f);**

**}**

**Console.ReadKey();**

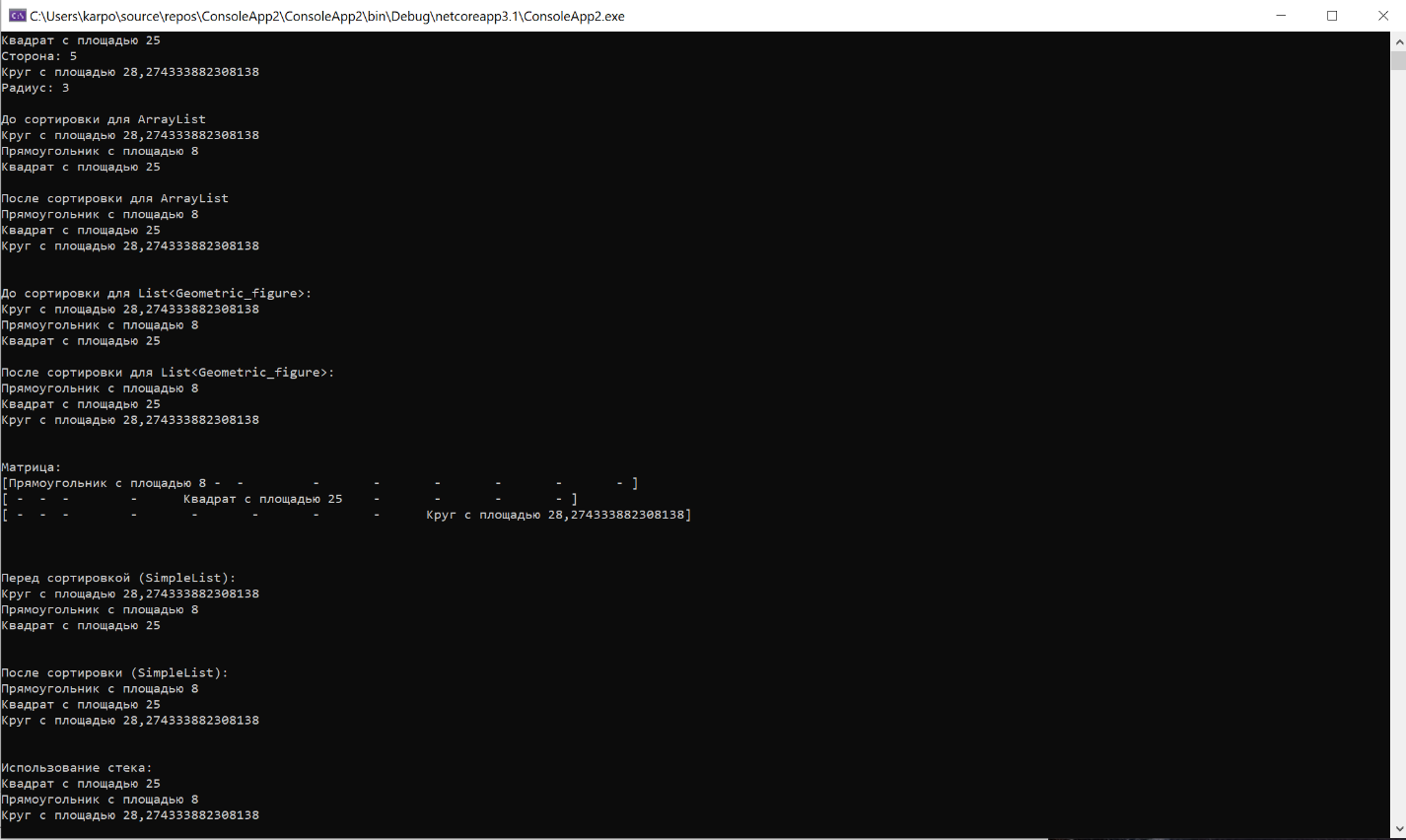
**}**

**}**

**}**

**}**

**Тест программы**

****

**Cсылка на репозиторий исходных кодов GitHub**

https://github.com/adidas1806/karpovigor