|  |  |
| --- | --- |
|  | **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**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Москальцов Никита Олегович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*фамилия, имя, отчество*

Группа: ИУ5 - 34

Название предмета**: Базовые компоненты интернет-технологий**

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** \_\_\_\_Москальцов Н.О.\_\_\_\_

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Гапанюк Ю.Е.**\_\_\_\_**

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

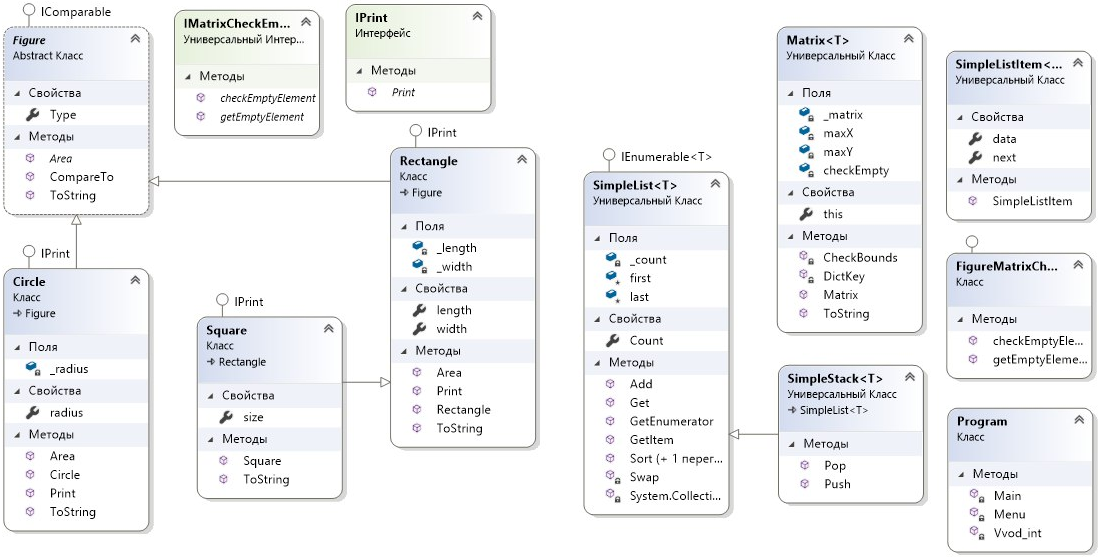
*2017 г.*

**Задание**

Разработать программу, реализующую работу с коллекциями.

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Для реализации возможности сортировки геометрических фигур для класса «Геометрическая фигура» добавить реализацию интерфейса IComparable. Сортировка производится по площади фигуры.
3. Создать коллекцию класса ArrayList. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.
4. Создать коллекцию класса List<Figure>. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.
5. 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.
6. Реализовать класс «SimpleStack» на основе односвязного списка. Класс SimpleStack наследуется от класса SimpleList (разобранного в пособии). Необходимо добавить в класс методы:
   * public void Push(T element) – добавление в стек;
   * public T Pop() – чтение с удалением из стека.
7. Пример работы класса Создать объекты классов «Прямоугольник», «Квадрат», «Круг».
8. SimpleStack реализовать на основе геометрических фигур.

**Диаграмма классов**



**Текст программы**

|  |
| --- |
| using System; |
|  | using System.Collections; |
|  | using System.Collections.Generic; |
|  | using System.Linq; |
|  | using System.Text; |
|  | using System.Threading.Tasks; |
|  |  |
|  | namespace Laba4 |
|  | { |
|  | interface IPrint |
|  | { |
|  | void Print(); |
|  | } |
|  | public interface IMatrixCheckEmpty<T> |
|  | { |
|  | // Возвращает пустой элемент |
|  | T getEmptyElement(); |
|  |  |
|  | //Проверка что элемент является пустым |
|  | bool checkEmptyElement(T element); |
|  | } |
|  | //Фигура |
|  | abstract class Figure : IComparable |
|  | { |
|  | //Тип фигуры |
|  | public string Type { get; protected set; } |
|  |  |
|  | //Абстрактный метод вычисления площади фигуры |
|  | public abstract double Area(); |
|  |  |
|  | //Метод вывода площади фигуры |
|  | public override string ToString() |
|  | { |
|  | return "Площадь фигуры: " + this.Area().ToString(); |
|  | } |
|  | // Сравнение для сортировки |
|  | // -1 - правый элемент больше левого |
|  | // 0 - один элемент равен другому |
|  | // 1 - правый элемент меньше левого |
|  | public int CompareTo(Object obj) |
|  | { |
|  | Figure p = (Figure)obj; |
|  | if (this.Area() < p.Area()) |
|  | return -1; |
|  | else if (this.Area() == p.Area()) |
|  | return 0; |
|  | else |
|  | return 1; |
|  | //(this.Area() > p.Area()) |
|  | } |
|  | } |
|  | //Прямоугольник |
|  | class Rectangle : Figure, IPrint |
|  | { |
|  | //Конструктор |
|  | public Rectangle(double w, double h) |
|  | { |
|  | this.Type = "Прямоугольник"; |
|  | this.width = w; |
|  | this.length = h; |
|  | } |
|  | //Длина |
|  | private double \_length = 0; |
|  | public double length |
|  | { |
|  | //возвращаемое значение |
|  | get |
|  | { |
|  | return \_length; |
|  | } |
|  | //установка значения, value - ключевое слово |
|  | set |
|  | { |
|  | \_length = value; |
|  | } |
|  | } |
|  | //Ширина |
|  | private double \_width = 0; |
|  | public double width |
|  | { |
|  | //возвращаемое значение |
|  | get |
|  | { |
|  | return \_width; |
|  | } |
|  | //установка значения, value - ключевое слово |
|  | set |
|  | { |
|  | \_width = value; |
|  | } |
|  | } |
|  | public override string ToString() |
|  | { |
|  | return this.Type + " со сторонами равными " + this.width + " и " |
|  | + this.length + " и площадью равной " + this.Area(); |
|  |  |
|  | } |
|  | public override double Area() |
|  | { |
|  | double Result = this.length \* this.width; |
|  | return Result; |
|  | } |
|  | public void Print() |
|  | { |
|  | Console.WriteLine(ToString()); |
|  | } |
|  | } |
|  | //Квадрат |
|  | class Square : Rectangle, IPrint |
|  | { |
|  | public double size { get; set; } |
|  | public Square(double s) : base(s, s) |
|  | { |
|  | this.Type = "Квадрат"; |
|  | this.size = s; |
|  | } |
|  | public override string ToString() |
|  | { |
|  | return this.Type + " со стороной " + this.size + " и площадью " + this.Area(); |
|  | } |
|  | } |
|  | //Круг |
|  | class Circle : Figure, IPrint |
|  | { |
|  | private double \_radius = 0; |
|  | public double radius |
|  | { |
|  | //возвращаемое значение |
|  | get |
|  | { |
|  | return \_radius; |
|  | } |
|  | //установка значения, value - ключевое слово |
|  | set |
|  | { |
|  | \_radius = value; |
|  | } |
|  | } |
|  | public Circle(double r) |
|  | { |
|  | this.radius = r; |
|  | this.Type = "Круг"; |
|  | } |
|  | public override double Area() |
|  | { |
|  | double Result = Math.PI \* this.radius \* this.radius; |
|  | return Result; |
|  | } |
|  | public void Print() |
|  | { |
|  | Console.WriteLine(ToString()); |
|  | } |
|  | public override string ToString() |
|  | { |
|  | return this.Type + " с радиусом " + this.radius + " и площадью " + this.Area(); |
|  | } |
|  | } |
|  |  |
|  | //Класс для разреженной проверки пустоты и ее задания |
|  | //наслудется от интерфейся с аналогичной задачей (для Figure) |
|  | class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure> |
|  | { |
|  | // В качестве пустого элемента возвращается null |
|  | public Figure getEmptyElement() |
|  | { |
|  | return null; |
|  | } |
|  | // Проверка на равенство null |
|  | public bool checkEmptyElement(Figure element) |
|  | { |
|  | bool Result = false; |
|  | if (element == null) |
|  | { |
|  | Result = true; |
|  | } |
|  | return Result; |
|  | } |
|  | } |
|  |  |
|  | //Класс матрицы |
|  | public class Matrix<T> |
|  | { |
|  | // Словарь для хранения значений |
|  | Dictionary<string, T> \_matrix = new Dictionary<string, T>(); |
|  |  |
|  | // Mаксимальное количество столбцов |
|  | int maxX; |
|  |  |
|  | // Mаксимальное количество строк |
|  | int maxY; |
|  |  |
|  | // Реализация интерфейса для проверки пустого элемента |
|  | IMatrixCheckEmpty<T> сheckEmpty; |
|  |  |
|  | // Конструктор |
|  | public Matrix(int px, int py, IMatrixCheckEmpty<T> сheckEmptyParam) |
|  | { |
|  | this.maxX = px; |
|  | this.maxY = py; |
|  | this.сheckEmpty = сheckEmptyParam; |
|  | } |
|  |  |
|  | // Индексатор для доступа к данных |
|  | public T this[int x, int y] |
|  | { |
|  | set |
|  | { |
|  | //Проверка выхода за границы матрицы |
|  | CheckBounds(x, y); |
|  |  |
|  | //Передача ключу значения координат элемента в матрице |
|  | string key = DictKey(x, y); |
|  |  |
|  | //Передача элемента в матрицу |
|  | this.\_matrix.Add(key, value); |
|  | } |
|  | get |
|  | { |
|  | CheckBounds(x, y); |
|  | string key = DictKey(x, y); |
|  | //Проверка принадлежности элемента матрице стандартным методом (из методички) |
|  | //Вопрос: какие ЕЩЁ такие стандартные методы есть у реализуемых нами объектов? |
|  | if (this.\_matrix.ContainsKey(key)) |
|  | { |
|  | return this.\_matrix[key]; |
|  | } |
|  | else |
|  | { |
|  | return this.сheckEmpty.getEmptyElement(); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | // Проверка границ |
|  | void CheckBounds(int x, int y) |
|  | { |
|  | if (x < 0 || x >= this.maxX) |
|  | { |
|  | //Выброс новых исключений |
|  | throw new ArgumentOutOfRangeException |
|  | ("x", "x=" + x + " выходит за границы"); |
|  | } |
|  | if (y < 0 || y >= this.maxY) |
|  | { |
|  | throw new ArgumentOutOfRangeException |
|  | ("y", "y=" + y + " выходит за границы"); |
|  | } |
|  | } |
|  |  |
|  | // Формирование ключа |
|  | string DictKey(int x, int y) |
|  | { |
|  | return x.ToString() + "\_" + y.ToString(); |
|  | } |
|  |  |
|  | // Приведение к строке для печати |
|  | 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++) |
|  | { |
|  | //Добавление разделителя-табуляции |
|  | if (i > 0) |
|  | { |
|  | b.Append("\t"); |
|  | } |
|  | //Если текущий элемент не пустой |
|  | if (!this.сheckEmpty.checkEmptyElement(this[i, j])) |
|  | { |
|  | //Добавить приведенный к строке текущий элемент |
|  | b.Append(this[i, j].ToString()); |
|  | } |
|  | else |
|  | { |
|  | //Иначе добавить признак пустого значения |
|  | b.Append(" - "); |
|  | } |
|  | } |
|  | b.Append("]\n"); |
|  | } |
|  | return b.ToString(); |
|  | } |
|  | } |
|  |  |
|  | //Класс элемента списка для создания класса стэка |
|  | public class SimpleListItem<T> |
|  | { |
|  | //Данные |
|  | public T data { set; get; } |
|  | //Следующий элемент |
|  | public SimpleListItem<T> next { set; get; } |
|  | //Конструктор |
|  | public SimpleListItem(T param) |
|  | { |
|  | this.data = param; |
|  | } |
|  | } |
|  | // Список |
|  | 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)) |
|  | { |
|  | 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() |
|  | { |
|  | //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; |
|  | } |
|  | this.Count--; |
|  | return Result; |
|  | } |
|  | } |
|  |  |
|  | //Основная программа |
|  | class Program |
|  | { |
|  | //Функция меню |
|  | static int Menu() |
|  | { |
|  | Console.Write("!!!Перед вами программа для работы с площадями геометрических фигур!!!\n"); |
|  | Console.Write("Выберите интересующую вас фигуру\n"); |
|  | Console.Write("================\n"); |
|  | Console.Write("1. Прямоугольник\n"); |
|  | Console.Write("2. Квадрат\n"); |
|  | Console.Write("3. Круг\n"); |
|  | Console.Write("4. Выход\n"); |
|  | Console.Write("================\n\n"); |
|  |  |
|  | int c; |
|  | c = Vvod\_int(); |
|  | return c; |
|  | } |
|  |  |
|  | //Функция ввода целого числа без ошибок |
|  | static int Vvod\_int() |
|  | { |
|  | bool result; |
|  | int c; |
|  | do |
|  | { |
|  | result = int.TryParse(Console.ReadLine(), out c); |
|  | if (result) |
|  | { |
|  | break; |
|  | } |
|  | else |
|  | { |
|  | Console.Write("Вы ввели не число! Пожалуйста, повторите ввод: "); |
|  | } |
|  | } while (true); |
|  | return c; |
|  | } |
|  |  |
|  | //Главная функция |
|  | static void Main(string[] args) |
|  | { |
|  | //Создание элементов фигур |
|  | //b = Vvod\_int(); ввод числа до верного ввода |
|  |  |
|  | Console.WriteLine("\nArrayList\n"); |
|  |  |
|  | Rectangle rec1 = new Rectangle(3, 4); |
|  | Square sq1 = new Square(5); |
|  | Circle c1 = new Circle(2); |
|  |  |
|  | //Коллекция класса ArrayList |
|  | ArrayList fig1 = new ArrayList(); |
|  | fig1.Add(rec1); |
|  | fig1.Add(sq1); |
|  | fig1.Add(c1); |
|  | //Печать до сортировки |
|  | Console.WriteLine("\nПеред сортировкой:"); |
|  | foreach (object o in fig1) |
|  | { |
|  | Console.WriteLine(o.ToString()); |
|  | } |
|  | fig1.Sort(); |
|  | //после сортировки |
|  | Console.WriteLine("\nПосле сортировки:"); |
|  | foreach (object o in fig1) |
|  | { |
|  | Console.WriteLine(o.ToString()); |
|  | } |
|  |  |
|  | Console.WriteLine("\nList\n"); |
|  |  |
|  | Rectangle rect = new Rectangle(5, 8); |
|  | Square square = new Square(4); |
|  | Circle circle = new Circle(6); |
|  |  |
|  | List<Figure> fig2 = new List<Figure>() |
|  | { |
|  | circle, square, rect |
|  | }; |
|  | //Печать до сортировки |
|  | Console.WriteLine("\nПеред сортировкой:"); |
|  | foreach (object o in fig1) |
|  | { |
|  | Console.WriteLine(o.ToString()); |
|  | } |
|  | fig2.Sort(); |
|  | //после сортировки |
|  | Console.WriteLine("\nПосле сортировки:"); |
|  | foreach (object o in fig2) |
|  | { |
|  | Console.WriteLine(o.ToString()); |
|  | } |
|  |  |
|  | //Матрица |
|  | Console.WriteLine("\nМатрица"); |
|  | Matrix<Figure> matrix = |
|  | new Matrix<Figure>(3, 3, new FigureMatrixCheckEmpty()); |
|  | matrix[0, 0] = rect; |
|  | matrix[1, 1] = square; |
|  | matrix[2, 2] = circle; |
|  | Console.WriteLine(matrix.ToString()); |
|  |  |
|  | //Нестандартный список |
|  | SimpleList<Figure> list = new SimpleList<Figure>(); |
|  | list.Add(circle); |
|  | list.Add(rect); |
|  | list.Add(square); |
|  | Console.WriteLine("\nПеред сортировкой:"); |
|  | foreach (var x in list) |
|  | { |
|  | Console.WriteLine(x); |
|  | } |
|  | //сортировка |
|  | list.Sort(); |
|  | Console.WriteLine("\nПосле сортировки:"); |
|  | foreach (var x in list) |
|  | { |
|  | Console.WriteLine(x); |
|  | } |
|  |  |
|  | //Стэк из фигур |
|  | Console.WriteLine("\nСтэк из фигур " + |
|  | "(видно, что печать и удаление соответственно идут начиная с последнего элемента)\n"); |
|  | SimpleStack<Figure> stack = new SimpleStack<Figure>(); |
|  | stack.Push(rect); |
|  | stack.Push(square); |
|  | stack.Push(circle); |
|  | while (stack.Count > 0) |
|  | { |
|  | Figure f = stack.Pop(); |
|  | Console.WriteLine(f); |
|  | } |
|  |  |
|  | Console.WriteLine("\nРабота программы завершена :)\n"); |
|  | Console.Read(); |
|  | } |
|  | } |
|  | } |

**Экранные формы с примерами выполнения программы**

