Лабораторная работа 1

Классы, свойства, индексаторы. Одномерные, прямоугольные и ступенчатые массивы

Требования к программе, общие для всех вариантов

Определить класс Person, который имеет

* закрытое поле типа string, в котором хранится имя;
* закрытое поле типа string, в котором хранится фамилия;
* закрытое поле типа System.DateTime для даты рождения.

В классе Person определить конструкторы:

* конструктор c тремя параметрами типа string, string, DateTime для инициализации всех полей класса;
* конструктор без параметров, инициализирующий все поля класса некоторыми значениями по умолчанию.

В классе Person определить свойства c методами get и set:

* свойство типа string для доступа к полю с именем;
* свойство типа string для доступа к полю с фамилией;
* свойство типа DateTime для доступа к полю с датой рождения;
* свойство типа int c методами get и set для получения информации(get) и изменения (set) года рождения в закрытом поле типа DateTime, в котором хранится дата рождения.

В классе Person определить

* перегруженную(override) версию виртуального метода string ToString() для формирования строки со значениями всех полей класса;
* виртуальный метод string ToShortString(), который возвращает строку, содержащую только имя и фамилию.

Сравнить время, необходимое для выполнения операций с элементами одномерного, двумерного прямоугольного и двумерного ступенчатого массивов с одинаковым числом элементов.

Для этого в методе Main() создать

* одномерный массив;
* двумерный прямоугольный массив;
* двумерный ступенчатый массив.

Тип элементов массивов зависит от варианта лабораторной работы. Массивы должны иметь одинаковое число элементов. Если число строк в двумерном прямоугольном массиве равно nrow, а число столбцов ncolumn, то одномерный массив должен содержать nrow\*ncolumn элементов, в двумерном ступенчатом массиве общее число элементов также должно быть равно nrow\*ncolumn. Значения nrow и ncolumn вводятся в процессе работы приложения в виде одной строки с разделителями. В приглашении, которое получает пользователь, должна быть информация о том, какие символы можно использовать как разделители, число разделителей должно быть больше 1. С помощью метода Split класса System.String приложение разбирает введенную пользователем текстовую строку с информацией о числе строк и числе столбцов двумерного массива и присваивает значения переменным, которые содержат значения nrow и ncolumn. В первой лабораторной работе не требуется обрабатывать ошибки ввода, предполагается, что пользователь правильно ввел данные.

Приложение распределяет память для всех массивов и инициализирует элементы массивов. Для инициализации элементов можно использовать конструктор без параметров.

Для всех элементов массивов выполняется одна и та же операция, например, присваивается значение одному из свойств, определенных для элементов массива. В лабораторной работе требуется сравнить время выполнения этой операции для одномерного, двумерного прямоугольного и двумерного ступенчатого массивов с одинаковым числом элементов.

Для измерения времени выполнения операций можно использовать свойство Environment.TickCount. Cтатическое свойство TickCount класса Environment имеет тип int, использует информацию системного таймера и содержит время в миллисекундах, которое прошло с момента перезагрузки компьютера.

Чтобы получить время выполнения некоторого блока кода, необходимо вызвать Environment.TickCount непосредственно перед блоком и сразу же после последнего оператора блока и взять разность значений.

В блоке кода, для которого измеряется время, не должно быть операций распределения памяти для массивов, инициализации элементов массивов и операций вывода данных на консоль. Блоки кода должны содержать только операции с элементами массива.

Вычисленные значения времени выполнения операций для трех типов массивов, а также число строк nrow и столбцов ncolumn выводятся на консоль. Вывод должен быть подписан, т.е. вывод должен содержать информацию о том, какому типу массива отвечает выведенное значение.

При работе в среде VisualStudio необходимо уметь

* средствами Solution Explorer добавить в проект новый класс;
* с помощью диаграммы классов (Class Diagram) добавить в класс методы, поля и свойства;
* с помощью диаграммы классов добавить в класс перегруженную (override) версию виртуального метода.

Вариант 1.

Требования к программе

Определить тип Education - перечисление(enum) со значениями Specialist, Вachelor, SecondEducation.

Определить класс Exam, который имеет три открытых автореализуемых свойства, доступных для чтения и записи:

* свойство типа string, в котором хранится название предмета;
* свойство типа int, в котором хранится оценка;
* свойство типа System.DateTime для даты экзамена.

В классе Exam определить:

* конструктор с параметрами типа string, int и DateTime для инициализации всех свойств класса;
* конструктор без параметров, инициализирующий все свойства класса некоторыми значениями по умолчанию;
* перегруженную(override) версию виртуального метода string ToString() для формирования строки со значениями всех свойств класса.

Определить класс Student, который имеет

* закрытое поле типа Person, в котором хранятся данные студента;
* закрытое поле типа Education для информации о форме обучения;
* закрытое поле типа int для номера группы;
* закрытое поле типа Exam \*+ для информации об экзаменах, которые сдал студент.

В классе Student определить конструкторы:

* конструктор c параметрами типа Person, Education, int для инициализации соответствующих полей класса;
* конструктор без параметров, инициализирующий поля класса значениями по умолчанию.

В классе Student определить свойства c методами get и set:

* свойство типа Person для доступа к полю с данными студента;
* свойство типа Education для доступа к полю с формой обучения;
* свойство типа int для доступа к полю с номером группы;
* свойство типа Exam \*+ для доступа к полю со списком экзаменов.

В классе Student определить

* свойство типа double (только с методом get), в котором вычисляется средний балл как среднее значение оценок в списке сданных экзаменов;
* индексатор булевского типа (только с методом get) с одним параметром типа Education; значение индексатора равно true, если значение поля с
* формой обучения студента совпадает со значением индекса, и false в противном случае;
* метод void AddExams ( params Exam [] ) для добавления элементов в список экзаменов;
* перегруженную версию виртуального метода string ToString() для формирования строки со значениями всех полей класса, включая список экзаменов;
* виртуальный метод string ToShortString(), который формирует строку со значениями всех полей класса без списка экзаменов, но со значением среднего балла.

В методе Main()

5. Создать один объект типа Student, преобразовать данные в текстовый вид с помощью метода ToShortString() и вывести данные.

6. Вывести значения индексатора для значений индекса Education.Specialist, Education.Bachelor и Education.SecondEducation.

7. Присвоить значения всем определенным в типе Student свойствам, преобразовать данные в текстовый вид с помощью метода ToString() и вывести данные.

8. C помощью метода AddExams(params Exam\*+ ) добавить элементы в список экзаменов и вывести данные объекта Student, используя метод ToString().

9. Сравнить время выполнения операций с элементами одномерного, двумерного прямоугольного и двумерного ступенчатого массивов с одинаковым числом элементов типа Exam.

Код программы (описание в комментариях):

1. Ниже представлен статический метод откуда начинается программа:

static void Main(string[] args)

{

//Создаем объект класса Student и выводим его данные на экран

var student = new Student();

Console.WriteLine("Студент: " + student.ToShortString());

//Получаем и выводим значения индексатора

var isSpecialist = student[EducationEnum.Specialist];

var isВachelor = student[EducationEnum.Вachelor];

var isSecondEducation = student[EducationEnum.SecondEducation];

Console.WriteLine

(@$"

{EducationEnum.Specialist} = {isSpecialist}

{EducationEnum.Вachelor} = {isВachelor}

{EducationEnum.SecondEducation} = {isSecondEducation}

");

//Заполняем с консоли объект Student с соответствующими проверками

Console.WriteLine("Заполните объект студента:");

Console.Write($"{nameof(student.PersonInfo.Name)} = ") ;

student.PersonInfo.Name = Console.ReadLine() ?? student.PersonInfo.Name;

Console.Write($"{nameof(student.PersonInfo.LastName)} = ");

student.PersonInfo.LastName = Console.ReadLine() ??

student.PersonInfo.LastName;

Console.Write($"{nameof(student.PersonInfo.DateOfBirth)} = ");

student.PersonInfo.DateOfBirth = DateTime.Parse(Console.ReadLine() ??

student.PersonInfo.DateOfBirth.ToShortDateString());

Console.Write($"{nameof(student.GroupNumber)} = ");

var validGroup = Int32.TryParse(Console.ReadLine(), out var group);

student.GroupNumber = validGroup ? group : 0;

Console.Write($"{nameof(student.Education)} = ");

student.Education = (EducationEnum)(Convert.ToInt16(Console.ReadLine()));

Console.WriteLine(student.ToString());

//Добавляем экзамены для студента и выводим на экран результат

Console.Write("Введите количество экзаменов для последующего их

автоматического заполнения: ");

var count = Convert.ToInt32(Console.ReadLine());

var exams = new Exam[count];

for(int i = 0; i < count; i++)

{

exams[i] = new Exam();

}

student.AddExams(exams);

Console.WriteLine(student.ToString());

//Создаем 3 массива(одномерный, двумерный и ступенчатый), предварительно

запросив количество строк и столбцов у пользователя

Console.Write("Введите кол-во строк и столбцов через запятую или пробел:

");

var data = Console.ReadLine().Split(new char[] { ',', ' ' });

var nrow = Convert.ToInt32(data[0]);

var ncolumn = Convert.ToInt32(data[1]);

var rectangleArray = new Exam[nrow, ncolumn];

var defaultArray = new Exam[nrow \* ncolumn];

var escaleraArray = new Exam[nrow \* ncolumn][];

//Заполняем массивы

for(int i = 0; i < nrow\*ncolumn; i++)

{

defaultArray[i] = new Exam();

}

for(int i = 0; i < nrow; i++)

{

for(int j = 0; j < ncolumn; j++)

{

rectangleArray[i, j] = new Exam();

}

}

for(int i = 0; i < nrow\*ncolumn; i++)

{

escaleraArray[i] = new Exam[1];

for (int j = 0; j < 1; j++)

escaleraArray[i][j] = new Exam();

}

//Считаем время работы каждого из них и выводим результат на консоль

var startRectangleArrayMs = Environment.TickCount64;

for (int i = 0; i < nrow; i++)

{

for (int j = 0; j < ncolumn; j++)

{

rectangleArray[i, j].Subject = ProgramConsts.TestSubjectName;

}

}

var endRectangleArrayMs = Environment.TickCount64;

Console.WriteLine($"Двумерный прямоугольный массив выполнил операцию за

{endRectangleArrayMs - startRectangleArrayMs} ms");

var startDefaultArrayMs = Environment.TickCount64;

for (int i = 0; i < nrow \* ncolumn; i++)

{

defaultArray[i].Subject = ProgramConsts.TestSubjectName;

}

var endDefaultArrayMs = Environment.TickCount64;

Console.WriteLine($"Одномерный массив выполнил операцию за

{endDefaultArrayMs - startDefaultArrayMs} ms");

var startEscaleraArrayMs = Environment.TickCount64;

for (int i = 0; i < nrow \* ncolumn; i++)

{

for (int j = 0; j < 1; j++)

escaleraArray[i][j].Subject = ProgramConsts.TestSubjectName;

}

var endEscaleraArrayMs = Environment.TickCount64;

Console.WriteLine($"Двумерный ступенчатый массив выполнил операцию за

{endEscaleraArrayMs - startEscaleraArrayMs} ms");

Console.ReadKey();

}

1. Определяем класс Person, который описывает общую модель человека со всеми необходимыми полями, свойствами и методами.

/// <summary>

/// Класс человека

/// </summary>

public class Person

{

/// <summary>

/// Имя

/// </summary>

private string \_name;

/// <summary>

/// Фамилия

/// </summary>

private string \_lastName;

/// <summary>

/// Дата рождения

/// </summary>

private DateTime \_dateOfBirth;

/// <summary>

/// Имя

/// </summary>

public string Name

{

get => \_name;

set => \_name = value;

}

/// <summary>

/// Фамилия

/// </summary>

public string LastName

{

get => \_lastName;

set => \_lastName = value;

}

/// <summary>

/// Дата рождения

/// </summary>

public DateTime DateOfBirth

{

get => \_dateOfBirth;

set => \_dateOfBirth = value;

}

/// <summary>

/// Год рождения

/// </summary>

public int CustomizeDateOfBirthYear

{

get

{

return \_dateOfBirth.Year;

}

set

{

\_dateOfBirth = \_dateOfBirth.AddYears(-

\_dateOfBirth.Year).AddYears(value);

}

}

/// <summary>

/// Конструктор заполняющий дефолтными значениями

/// </summary>

public Person()

{

\_name = ProgramConsts.DefaultName;

\_lastName = ProgramConsts.DefaultLastName;

\_dateOfBirth = DateTime.Now.AddYears(-20);

}

/// <summary>

/// Конструктор с параметрами

/// </summary>

public Person(string name, string lastName, DateTime datetime)

{

\_name = name;

\_lastName = lastName;

\_dateOfBirth = datetime;

}

/// <summary>

/// Получение строки со всеми полями класса

/// </summary>

/// <returns></returns>

public override string ToString()

{

return $"Имя: {\_name}, Фамилия: {\_lastName}, Дата рождения

{\_dateOfBirth}";

}

/// <summary>

/// Получение строки с именем и фамилией

/// </summary>

/// <returns></returns>

public virtual string ToShortString()

{

return $"Имя: {\_name}, Фамилия: {\_lastName}";

}

/// <summary>

/// Сравнение оьъектов

/// </summary>

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

/// <returns></returns>

public override bool Equals(object? obj)

{

if (Object.ReferenceEquals(this, obj))

return true;

if (this.GetType() != obj.GetType())

return false;

var person = obj as Person;

return person != null && this.Name == person.Name && LastName ==

person.LastName && DateOfBirth == person.DateOfBirth;

}

/// <summary>

/// Проверка на равенство

/// </summary>

/// <param name="person1"></param>

/// <param name="person2"></param>

/// <returns></returns>

public static bool operator == (Person person1, Person person2)

{

if(person1 is null)

{

if(person2 is null)

{

return true;

}

return false;

}

return person1.Equals(person2);

}

/// <summary>

/// Проверка на неравенство

/// </summary>

/// <param name="person1"></param>

/// <param name="person2"></param>

/// <returns></returns>

public static bool operator != (Person person1, Person person2)

{

return !(person1 == person2);

}

/// <summary>

/// Получение хэш кода

/// </summary>

/// <returns></returns>

public override int GetHashCode()

{

return (Name, LastName, DateOfBirth).GetHashCode();

}}

1. Определяем класс Student, описывающий конкретно студента и ссылающийся на класс Person через поле \_personInfo.

public class Student

{

/// <summary>

/// Информация о студенте

/// </summary>

private Person \_personInfo;

/// <summary>

/// Образование

/// </summary>

private EducationEnum \_education;

/// <summary>

/// Номер группы

/// </summary>

private int \_groupNumber;

/// <summary>

/// Экзамены

/// </summary>

private Exam[] \_exams;

/// <summary>

/// Информация о студенте

/// </summary>

public Person PersonInfo

{

get => \_personInfo;

set => \_personInfo = value;

}

/// <summary>

/// Образование

/// </summary>

public EducationEnum Education

{

get => \_education;

set => \_education = value;

}

/// <summary>

/// Номер группы

/// </summary>

public int GroupNumber

{

get => \_groupNumber;

set => \_groupNumber = value;

}

/// <summary>

/// Экзамены

/// </summary>

public Exam[] Exams

{

get => \_exams;

set => \_exams = value;

}

/// <summary>

/// Средняя оценка

/// </summary>

public double AvgGrade { get => \_exams == null ? 0 : \_exams.Average(c =>

c.Grade); }

/// <summary>

/// Индексатор

/// </summary>

/// <param name="index"></param>

/// <returns></returns>

public bool this[EducationEnum education]

{

get => \_education == education ? true : false;

}

/// <summary>

/// Конструктор с параметрами

/// </summary>

/// <param name="personInfo"></param>

/// <param name="education"></param>

/// <param name="groupNumber"></param>

public Student(Person personInfo, EducationEnum education, int groupNumber)

{

\_personInfo = personInfo;

\_education = education;

\_groupNumber = groupNumber;

}

/// <summary>

/// Конструктор без параметров

/// </summary>

public Student()

{

\_personInfo = new Person();

\_groupNumber = new Random().Next(100, 401);

\_education = EducationEnum.Вachelor;

}

/// <summary>

/// Добавление экзаменов

/// </summary>

/// <param name="exams"></param>

public void AddExams(params Exam[] exams)

{

\_exams = exams;

}

/// <summary>

/// Получение строки со всеми полями класса

/// </summary>

/// <returns></returns>

public override string ToString()

{

var str = $"{\_personInfo.ToString()}\nОбразование: {\_education}, Группа:

{\_groupNumber}\n";

if (\_exams != null && \_exams.Count() > 0)

str += string.Join(",\n", \_exams.Select(c =>

c.ToString()).ToList());

return str;

}

/// <summary>

/// Получение строки с именем и фамилией

/// </summary>

/// <returns></returns>

public virtual string ToShortString()

{

return $"{\_personInfo.ToString()}\nОбразование: {\_education}, Группа:

{\_groupNumber}, Средний балл: {AvgGrade}\n";

}

}

1. Для того чтобы задать студенту тип образования используем ключевое слово enum и создаем перечисление.

public enum EducationEnum

{

/// <summary>

/// Специалист

/// </summary>

Specialist,

/// <summary>

/// Балаклавр

/// </summary>

Вachelor,

/// <summary>

/// Второе образование

/// </summary>

SecondEducation

}

1. У каждого студента могут быть экзамены, их список мы добавили в класс Student. Сам класс Exam класс представлен ниже.

public class Exam

{

/// <summary>

/// Предмет

/// </summary>

public string Subject { get; set; }

/// <summary>

/// Оценка

/// </summary>

public int Grade { get; set; }

/// <summary>

/// Начало экзамена

/// </summary>

public DateTime DateStart { get; set; }

public Exam()

{

Subject = StringExtension.GetRandom(7);

Grade = new Random().Next(1, 6);

DateStart = DateTime.UtcNow;

}

public Exam(string subject, int grade, DateTime date)

{

Subject = subject;

Grade = grade;

DateStart = date;

}

/// <summary>

/// Получение строки со всеми полями

/// </summary>

/// <returns></returns>

public override string ToString()

{

return $"Предмет: {Subject}, Оценка: {Grade}, Дата начала: {DateStart}";

}

}

Консольный вывод:

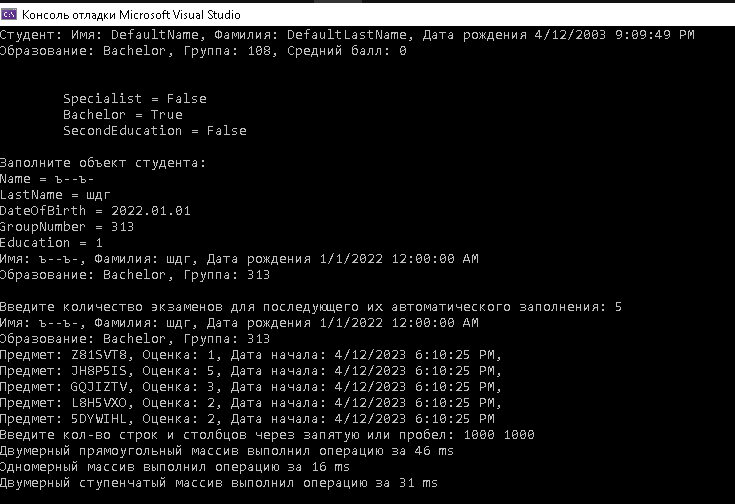


Рис. 1 Результат выполнения лабораторной работы №1