Лабораторная работа 5. Варианты первого уровня

Классы для работы с файлами. Сериализация.

Требования к программе, общие для всех вариантов

Определить новые версии классов Student (вариант 1), Magazine (вариант 2) и ResearchTeam (вариант 3). В сформулированных ниже требованиях для этих классов использовано общее обозначение T.

В новые версии классов добавить экземплярные методы

• T DeepCopy() для создания полной копии объекта с использованием сериализации;

• bool Save(string filename) для сохранения данных объекта в файле с помощью сериализации;

• bool Load(string filename) для инициализации объекта данными из файла с помощью десериализации;

• bool AddFromConsole() для добавления в один из списков класса нового элемента, данные для которого вводятся с консоли;

и статические методы

• static bool Save(string filename, T obj) для сохранения объекта в файле с помощью сериализации;

• static bool Load(string filename, T obj) для восстановления объекта из файла с помощью десериализации.

В экземплярном методе T DeepCopy() вызывающий объект сериализуется в поток MemoryStream. Метод возвращает восстановленный при десериализации объект, который представляет собой полную копию исходного объекта.

Экземплярный метод bool Save(string filename) сериализует все данные вызывающего объекта в файл с именем filename. Если файл с именем filename существует, приложение его перезаписывает. Если такого файла нет, приложение его создает. Метод возвращает значение true, если сериализация завершилась успешно, и значение false в противном случае.

Экземплярный метод bool Load(string filename) десериализует данные из файла с именем filename и использует их для инициализации вызывающего 61 объекта. Метод возвращает значение true, если инициализация завершилась успешно. Если полностью выполнить инициализацию объекта не удалось, исходные данные объекта должны остаться без изменения. В этом случае метод возвращает значение false.

Статические методы bool Save(string filename, T obj) и bool Load(string filename, T obj) получают через параметры имя файла и ссылку на объект, для которого выполняется сериализация или восстановление. Методы возвращают значение true, если сериализация/инициализация завершилась успешно, и значение false в противном случае. Если полностью выполнить инициализацию объекта не удалось, исходные данные объекта должны остаться без изменения.

Во всех реализациях методов сохранения/восстановления данных из файла операции открытия файла, сериализации и десериализации данных должны находиться в блоках try-catch-finally.

В методе bool AddFromConsole() для добавления нового элемента в один из списков класса T

• пользователь получает приглашение ввести данные в виде одной строки символов с разделителями; приглашение содержит описание формата строки ввода, в том числе информацию о том, какие символы можно использовать в качестве разделителей;

• выполняется разбор данных; операции преобразования данных, которые могут бросить исключение, должны находиться в блоке trycatch;

• если разбор введенных данных был завершен успешно, в список добавляется новый элемент и метод возвращает значение true; в противном случае пользователь получает сообщение о том, что при вводе были допущены ошибки и возвращаемое значение метода равно false.

Вариант 1.

Требования к программе

В варианте 1 элементы, данные для которых вводятся с консоли, добавляются в список экзаменов System.Collections.Generic.List. Вводятся название предмета, оценка и дата экзамена.

В методе Main()

1. Создать объект типа T с непустым списком элементов, для которого предусмотрен ввод данных с консоли. Создать полную копию объекта с помощью метода, использующего сериализацию, и вывести исходный объект и его копию.

2. Предложить пользователю ввести имя файла:

• если файла с введенным именем нет, приложение должно сообщить об этом и создать файл;

• если файл существует, вызвать метод Load(string filename) для инициализации объекта T данными из файла.

3. Вывести объект T.

4. Для этого же объекта T сначала вызвать метод AddFromConsole(), затем метод Save(string filename). Вывести объект T.

5. Вызвать последовательно

• статический метод Load( string filename, T obj), передав как параметры ссылку на тот же самый объект T и введенное ранее имя файла;

• метод AddFromConsole();

• статический метод Save (string filename, T obj).

6. Вывести объект T.

Приложение должно работать в режиме накопления. Если выбирается один и тот же файл для записи, и пользователь вводит данные без ошибок, при каждом следующем выполнении приложения к списку добавляются два новых элемента. Приложение должно обрабатывать все исключения, которые могут возникнуть из-за ошибок при вводе данных. Независимо от того, корректно были введены данные или при вводе были допущены ошибки, все файловые потоки должны быть закрыты.

Код программы (описание в комментариях):

1. Ниже преведен класс Program и метод Main откуда начинается программа.

public class Program

{

static void Main(string[] args)

{

//Создание студента

var student = new Student();

//Добавление эезаменов

student.AddExams(new List<Exam>

{

new Exam(),

new Exam()

});

//Добавление экзаменов через консоль

student.AddFromConsole();

Console.WriteLine();

//Создание полной копии

var copy = student.DeepCopy();

//Вывод информации

Console.WriteLine("Оригинал:");

Console.WriteLine(student.ToString());

Console.WriteLine("\nКопия:");

Console.WriteLine(copy.ToString());

Console.WriteLine();

//Запрос на ввод пути к файлу

Console.Write("\nВведите имя файла: ");

var filename = Console.ReadLine();

//Проверка на существование

if (File.Exists(filename))

{

//Создание файла

Console.WriteLine("Файла не существует");

File.Create(filename);

}

else

{

//Загрузка объекта из файла

student.Load(filename);

}

//Вывод информации

Console.WriteLine();

Console.WriteLine(student.ToString());

//Добавление экзаменов через консоль

student.AddFromConsole();

student.Save(filename);

//Вывод на экран

Console.WriteLine();

Console.WriteLine(student.ToString());

//Загрузка объекта из файла

Student.Load(filename, student);

//Добавление экзаменов

student.AddFromConsole();

//Сохранение в файл

Student.Save(filename, student);

//Вывод информации

Console.WriteLine();

Console.WriteLine(student.ToString());

}

}

1. Определяем новую версию класса Student.

[Serializable]

public class Student : Person, IDateAndCopy

{

/// <summary>

/// Образование

/// </summary>

private EducationEnum \_education;

/// <summary>

/// Номер группы

/// </summary>

private int \_groupNumber;

/// <summary>

/// Экзамены

/// </summary>

private List<Exam> \_exams;

/// <summary>

/// Зачеты

/// </summary>

private List<Test> \_tests;

/// <summary>

/// Образование

/// </summary>

public EducationEnum Education

{

get => \_education;

set => \_education = value;

}

/// <summary>

/// Номер группы

/// </summary>

public int GroupNumber

{

get

{

return \_groupNumber;

}

set

{

if (value <= 100 || value > 599)

throw new Exception("Значение не соответствует условию > 100 и

<= 599");

\_groupNumber = value;

}

}

/// <summary>

/// Экзамены

/// </summary>

public List<Exam> Exams

{

get => \_exams;

set => \_exams = value;

}

/// <summary>

/// Тесты

/// </summary>

public List<Test> Tests

{

get => \_tests;

set => \_tests = value;

}

/// <summary>

/// Средняя оценка

/// </summary>

public double AvgGrade { get => \_exams == null ? 0 :

\_exams.ToArray().Average(c => ((Exam)c).Grade); }

/// <summary>

/// Дата

/// </summary>

public DateTime Date { get; set; }

/// <summary>

/// Индексатор

/// </summary>

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

/// <returns></returns>

public bool this[EducationEnum education]

{

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

}

public Person Person

{

get

{

return (Person)base.MemberwiseClone();

}

set

{

this.Name = value.Name;

this.LastName = value.LastName;

this.DateOfBirth = value.DateOfBirth;

}

}

/// <summary>

/// Конструктор с параметрами

/// </summary>

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

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

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

public Student(Person person, EducationEnum education, int groupNumber) :

base(person.Name, person.LastName, person.DateOfBirth)

{

\_education = education;

\_groupNumber = groupNumber;

}

/// <summary>

/// Конструктор без параметров

/// </summary>

public Student() : base()

{

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

\_education = (EducationEnum)new Random().Next(0, 3);

}

/// <summary>

/// Добавление экзаменов

/// </summary>

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

public void AddExams(List<Exam> exams)

{

if (\_exams == null)

\_exams = exams;

else

\_exams.AddRange(exams);

}

/// <summary>

/// Добавление тестов

/// </summary>

/// <param name="tests"></param>

public void AddTests(List<Test> tests)

{

if (\_tests == null)

\_tests = tests;

else

\_tests.AddRange(tests);

}

/// <summary>

/// Получение строки со всеми полями класса

/// </summary>

/// <returns></returns>

public override string ToString()

{

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

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

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

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

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

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

str += "\n" + string.Join(",\n", \_tests.ToArray().Select(c =>

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

return str;

}

/// <summary>

/// Получение строки с именем и фамилией

/// </summary>

/// <returns></returns>

public virtual string ToShortString()

{

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

{\_groupNumber}, Средний балл: {AvgGrade}, Кол-во экзаменов:

{\_exams.Count}, Кол-во зачетов: {\_tests.Count}\n";

}

/// <summary>

/// Получение полной копии

/// </summary>

/// <returns></returns>

public override Student DeepCopy()

{

using (MemoryStream stream = new MemoryStream())

{

BinaryFormatter formatter = new BinaryFormatter();

formatter.Serialize(stream, this);

stream.Seek(0, SeekOrigin.Begin);

var copiedObj = (Student)formatter.Deserialize(stream);

return copiedObj;

}

}

/// <summary>

/// Сохранение объекта

/// </summary>

/// <param name="filename"></param>

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

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

/// <returns></returns>

public static bool Save(string filename, Student student)

{

try

{

using (FileStream fs = new FileStream(filename, FileMode.Create))

{

BinaryFormatter formatter = new BinaryFormatter();

formatter.Serialize(fs, student);

return true;

}

}

catch (Exception ex)

{

return false;

}

}

/// <summary>

/// Сохранение

/// </summary>

/// <returns></returns>

public bool Save(string filename)

{

try

{

using (FileStream fs = new FileStream(filename, FileMode.Create))

{

BinaryFormatter formatter = new BinaryFormatter();

formatter.Serialize(fs, this);

return true;

}

}

catch (Exception ex)

{

return false;

}

}

/// <summary>

/// Выгрузка объекта

/// </summary>

/// <param name="filename"></param>

/// <param name="student"></param>

/// <returns></returns>

public static bool Load(string filename, Student student)

{

try

{

if (File.Exists(filename))

{

using (FileStream fs = new FileStream(filename, FileMode.Open))

{

BinaryFormatter formatter = new BinaryFormatter();

var deserilizeStudent = (Student)formatter.Deserialize(fs);

student = deserilizeStudent;

return true;

}

}

else

{

return false;

}

}

catch (Exception ex)

{

return false;

}

}

/// <summary>

/// Выгрузка объекта

/// </summary>

/// <param name="filename"></param>

/// <returns></returns>

public bool Load(string filename)

{

try

{

if (File.Exists(filename))

{

using (FileStream fs = new FileStream(filename, FileMode.Open))

{

BinaryFormatter formatter = new BinaryFormatter();

var deserilizeStudent = (Student)formatter.Deserialize(fs);

this.Name = deserilizeStudent.Name;

this.Person = deserilizeStudent.Person;

this.CustomizeDateOfBirthYear =

deserilizeStudent.CustomizeDateOfBirthYear;

this.Education = deserilizeStudent.Education;

this.Tests = deserilizeStudent.Tests;

this.Exams = deserilizeStudent.Exams;

this.GroupNumber = deserilizeStudent.GroupNumber;

this.LastName = deserilizeStudent.LastName;

return true;

}

}

else

{

return false;

}

}

catch (Exception ex)

{

return false;

}

}

/// <summary>

/// Итератор

/// </summary>

/// <returns></returns>

public IEnumerable GetEnumerator(double? minGrade)

{

if (minGrade == null)

{

foreach (var item in Exams)

{

yield return item;

}

foreach(var item in Tests)

{

yield return item;

}

}

else

foreach (var item in Exams)

{

if (((Exam)item).Grade > minGrade)

yield return (Exam)item;

}

}

/// <summary>

/// Консольное добавление

/// </summary>

/// <returns></returns>

public bool AddFromConsole()

{

Console.WriteLine($"Введите данные в виде одной строки для добавления

элемента. Разделителями служат запятые. Все данные указываются в

определенном проядке(Название предмета, Оценка, Дата)\n");

var insert = Console.ReadLine();

try

{

var data = insert.Split(',');

var subject = String.IsNullOrWhiteSpace(data[0]) ?

StringExtension.GetRandom(7) : data[0];

var grade = Int32.TryParse(data[1], out var groupResult) ?

groupResult : 0;

var date = DateTime.TryParse(data[2], out var dataResult) ?

dataResult : DateTime.MinValue;

var student = new Exam(subject, grade, date);

this.Exams.Add(student);

return true;

}

catch (Exception ex)

{

return false;

}

}

}

1. Воссоздаем класс Exam из прежних лабораторных работ

[Serializable]

public class Exam : IDateAndCopy

{

/// <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; }

/// <summary>

/// Дата

/// </summary>

public DateTime Date { 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}";

}

/// <summary>

/// Полуение полнй копии

/// </summary>

/// <returns></returns>

/// <exception cref="NotImplementedException"></exception>

public object DeepCopy()

{

return new Exam(Subject, Grade, DateStart);

}

}

1. Воссоздаем класс Test из прежних лабораторных работ

[Serializable]

public class Test : IDateAndCopy

{

/// <summary>

/// Название

/// </summary>

public string Name { get; set; }

/// <summary>

/// Сдан ли

/// </summary>

public bool IsPassed { get; set; }

public DateTime Date { get; set; }

public Test(string name, bool isPassed)

{

Name = name;

IsPassed = isPassed;

}

public Test()

{

Name = StringExtension.GetRandom(10);

IsPassed = false;

}

public override string ToString()

{

return $"Имя: {Name}, Зачет: {IsPassed}";

}

public object DeepCopy()

{

return new Test(Name, IsPassed);

}

}

Консольный вывод:

