Лабораторная работа 2

Наследование. Исключения. Интерфейсы. Итераторы и блоки итераторов.

Требования к программе, общие для всех вариантов:

В классе Person из лабораторной работы 1 и в классах, дополнительно указанных в вариантах, надо

• переопределить (override) виртуальный метод bool Equals (object obj);

• определить операции == и != ;

• переопределить виртуальный метод int GetHashCode();

Реализация виртуального метода bool Equals (object obj) в классе System.Object определяет равенство объектов как равенство ссылок на объекты. Некоторые классы из базовой библиотеки BCL переопределяют метод Equals(). В классе System.String этот метод переопределен так, что равными считаются строки, которые совпадают посимвольно. Реализация метода Equals() в структурном типе DateTime равенство объектов DateTime определяет как равенство значений.

В лабораторной работе требуется переопределить метод Equals так, чтобы объекты считались равными, если равны все данные объектов. Для класса Person это означает, что равны даты рождения и посимвольно совпадают строки с именем и фамилией.

Определение операций == и != должно быть согласовано с переопределенным методом Equals, т.е. критерии, по которым проверяется равенство объектов в методе Equals, должны использоваться и при проверке равенства объектов в операциях == и !=.

Переопределение виртуального метода int GetHashCode() также должно быть согласовано с операциями == и !=. Виртуальный метод GetHashCode() используется некоторыми классами базовой библиотеки, например, коллекциями-словарями. Классы базовой библиотеки, вызывающие метод GetHashCode() из пользовательского типа, предполагают, что равным объектам отвечают равные значения хэш-кодов.

Поэтому в случае, когда под равенством объектов понимается совпадение данных (а не ссылок), реализация метода 15 GetHashCode() должна для объектов с совпадающими данными возвращать равные значения хэш-кодов. В классах, указанных в вариантах лабораторной работы, требуется определить метод object DeepCopy() для создания полной копии объекта. Определенные в некоторых классах базовой библиотеки методы Clone() и Copy() создают ограниченную (shallow) копию объекта – при копировании объекта копии создаются только для полей структурных типов, для полей ссылочных типов копируются только ссылки. В результате в ограниченной копии объекта поля-ссылки указывают на те же объекты, что и в исходном объекте.

Метод DeepCopy() должен создать полные копии всех объектов, ссылки на которые содержат поля типа. После создания полная копия не зависит от исходного объекта - изменение любого поля или свойства исходного объекта не должно приводить к изменению копии.

При реализации метода DeepCopy() в классе, который имеет поле типа System.Collections.ArrayList, следует иметь в виду, что определенные в классе ArrayList конструктор ArrayList(ICollection) и метод Clone() при создании копии коллекции, состоящей из элементов ссылочных типов, копируют только ссылки.

Метод DeepCopy() должен создать как копии элементов коллекции ArrayList, так и полные копии объектов, на которые ссылаются элементы коллекции. Для типов, содержащих коллекции, реализация метода DeepCopy() упрощается, если в типах элементов коллекций также определить метод DeepCopy().

Вариант 1.

Требования к программе

Определить интерфейс

interface IDateAndCopy

{

object DeepCopy();

DateTime Date { get; set; }

}

Определить новые версии классов Exam, Person и Student из лабораторной работы 1. В классы Exam, Person и Student добавить реализацию интерфейса IDateAndCopy. Новую версию класса Student определить как класс, производный от класса Person.

Все поля новой версии класса Person определить с доступом protected, сохранить все свойства, определенные в первой версии класса.

В новой версии класса Person дополнительно

• переопределить метод virtial bool Equals (object obj) и определить операции == и != так, чтобы равенство объектов типа Person трактовалось как совпадение всех данных объектов, а не ссылок на объекты Person;

• переопределить виртуальный метод int GetHashCode();

• определить виртуальный метод object DeepCopy();

• реализовать интерфейс IDateAndCopy.

Определить класс Test, который имеет два открытых автореализуемых свойства, доступных для чтения и записи:

• свойство типа string, в котором хранится название предмета;

• свойство типа bool для информации о том, сдан зачет или нет.

В классе Test определить:

• конструктор c параметрами типа string и bool для инициализации 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.

Класс Student определить как производный от класса Person.

Новая версия класса Student имеет следующие поля:

• закрытое поле типа Education для информации о форме обучения;

• закрытое поле типа int для номера группы;

• закрытое поле типа System.Collections.ArrayList, в котором хранится список зачетов (объекты типа Test);

• закрытое поле типа System.Collections.ArrayList для списка экзаменов (объекты типа Exam).

Код следующих конструкторов, методов и свойств из старой версии класса Student необходимо изменить с учетом того, что часть полей класса перемещена в базовый класс Person, и в новой версии класса Student список экзаменов хранится в коллекции System.Collections.ArrayList:

• конструктор c параметрами типа Person, Education, int для инициализации соответствующих полей класса;

• конструктор без параметров для инициализации по умолчанию;

• свойство типа Person; метод get свойства возвращает объект типа Person, данные которого совпадают с данными подобъекта базового класса, метод set присваивает значения полям из подобъекта базового класса;

• свойство типа double ( только с методом get), в котором вычисляется средний балл как среднее значение оценок в списке сданных экзаменов;

• свойство типа System.Collections.ArrayList с методами get и set для доступа к полю со списком экзаменов;

• метод void AddExams ( params Exam[] ) для добавления элементов в список экзаменов;

• перегруженная версия виртуального метода string ToString() для формирования строки со значениями всех полей класса, включая список зачетов и экзаменов;

• виртуальный метод string ToShortString(), который формирует строку со значениями всех полей класса без списка зачетов и экзаменов, но со значением среднего балла.

Дополнительно в новой версии класса Student

• определить перегруженную версию виртуального метода object DeepCopy();

• реализовать интерфейс IDateAndCopy;

• определить свойство типа int с методами get и set для доступа к полю с номером группы. В методе set бросить исключение, если присваиваемое значение меньше или равно 100 или больше 599. При создании объектаисключения использовать один из определенных в библиотеке CLR классовисключений, инициализировать объект-исключение с помощью конструктора с параметром типа string, в сообщении передать информацию о допустимых границах для значения свойства.

В новой версии класса Student определить

• итератор для последовательного перебора всех элементов (объектов типа object) из списков зачетов и экзаменов (объединение);

• итератор c параметром для перебора экзаменов (объектов типа Exam) с оценкой больше заданного значения.

В методе Main()

1. Создать два объекта типа Person с совпадающими данными и проверить, что ссылки на объекты не равны, а объекты равны, вывести значения хэшкодов для объектов.

2. Создать объект типа Student, добавить элементы в список экзаменов и зачетов, вывести данные объекта Student.

3. Вывести значение свойства типа Person для объекта типа Student.

4. С помощью метода DeepCopy() создать полную копию объекта Student. Изменить данные в исходном объекте Student и вывести копию и исходный объект, полная копия исходного объекта должна остаться без изменений.

5. В блоке try/catch присвоить свойству с номером группы некорректное значение, в обработчике исключения вывести сообщение, переданное через объект-исключение.

6. С помощью оператора foreach для итератора, определенного в классе Student, вывести список всех зачетов и экзаменов.

7. С помощью оператора foreach для итератора с параметром, определенного в классе Student, вывести список всех экзаменов с оценкой выше 3.

Код программы (описание в комментариях):

1. Ниже приведен метод Main с которого начинается работа программы.

static void Main(string[] args)

{

//Добавление двух человвек

var person1 = new Person(ProgramConsts.DefaultName,

ProgramConsts.DefaultLastName, new DateTime(2022, 1, 1));

var person2 = new Person(ProgramConsts.DefaultName,

ProgramConsts.DefaultLastName, new DateTime(2022,1,1));

//Вывод информации

Console.WriteLine($"Равенство объектов: {person1.Equals(person2)}");

Console.WriteLine($"Равенство ссылок: {Object.ReferenceEquals(person1,

person2)}");

Console.WriteLine($"Хэш коды:

\n{person1.GetHashCode()}\n{person2.GetHashCode()}");

//Добавление нового студента и двух списков

var newSudent = new Student();

ArrayList exams = new ArrayList();

ArrayList tests = new ArrayList();

//Заполнение списков

for(int i = 0; i < 10; i++)

{

exams.Add(new Exam());

tests.Add(new Test());

}

//Добавление экзаменов и зачетов

newSudent.AddExams(exams);

newSudent.AddTests(tests);

//Вывод ифнормации

Console.WriteLine(newSudent.ToString());

Console.WriteLine("\nЗначения person из student: " +

((Person)newSudent).ToString());

try

{

//Копирование студента

var copyStudent = newSudent.DeepCopy();

//Изменение оригинала

newSudent.Name = "UpdatedName";

newSudent.LastName = "UpdatedLastName";

newSudent.GroupNumber = 200;

newSudent.Education =

Practice.Models.Enums.EducationEnum.SecondEducation;

newSudent.Date = new DateTime(2023,1,1);

newSudent.Exams[0] = new Exam("TestExam", 1, DateTime.Now);

newSudent.Tests[0] = new Test("TestTest", true);

Console.WriteLine($"Оригинал: \n{newSudent.ToString()}");

Console.WriteLine($"Копия: \n{copyStudent.ToString()}");

newSudent.GroupNumber = 1;

}

catch(Exception ex)

{

//Вывод ошибки

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(ex.Message);

Console.ResetColor();

}

//Перечисление экзаменов и зачетов

Console.WriteLine($"\nВывод с использованием итератора:");

foreach(var value in newSudent.GetEnumerator(null))

{

Console.WriteLine(value.ToString());

}

//Вывод экзаменов у которых оценка выше 3-х

Console.WriteLine($"\nВывод с использованием итератора с параметром:");

foreach (var value in newSudent.GetEnumerator(3))

{

Console.WriteLine(value.ToString());

}

}

1. Создаем новую версию класса Person из первой лабораторной работы.

/// <summary>

/// Класс человека

/// </summary>

public class Person : IDateAndCopy

{

/// <summary>

/// Имя

/// </summary>

protected string \_name;

/// <summary>

/// Фамилия

/// </summary>

protected string \_lastName;

/// <summary>

/// Имя

/// </summary>

public string Name

{

get => \_name;

set => \_name = value;

}

/// <summary>

/// Фамилия

/// </summary>

public string LastName

{

get => \_lastName;

set => \_lastName = value;

}

/// <summary>

/// Год рождения

/// </summary>

public int CustomizeDateOfBirthYear

{

get

{

return Date.Year;

}

set

{

Date = Date.AddYears(-Date.Year).AddYears(value);

}

}

/// <summary>

/// Дата рождения

/// </summary>

public DateTime Date { get; set; }

/// <summary>

/// Конструктор заполняющий дефолтными значениями

/// </summary>

public Person()

{

\_name = ProgramConsts.DefaultName;

\_lastName = ProgramConsts.DefaultLastName;

Date = DateTime.Now.AddYears(-20);

}

/// <summary>

/// Конструктор с параметрами

/// </summary>

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

{

\_name = name;

\_lastName = lastName;

Date = datetime;

}

/// <summary>

/// Получение строки со всеми полями класса

/// </summary>

/// <returns></returns>

public override string ToString()

{

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

}

/// <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 = (Person?)obj;

return this.Name == person.Name && LastName == person.LastName && Date

== person.Date;

}

/// <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, Date).GetHashCode();

}

/// <summary>

/// Создание полной копии

/// </summary>

/// <returns></returns>

public virtual object DeepCopy()

{

return new Person(this.Name, this.LastName, this.Date);

}

}

1. Создаем новую версию класса Student

public class Student : Person, IDateAndCopy

{

/// <summary>

/// Образование

/// </summary>

private EducationEnum \_education;

/// <summary>

/// Номер группы

/// </summary>

private int \_groupNumber;

/// <summary>

/// Экзамены

/// </summary>

private ArrayList \_exams;

/// <summary>

/// Зачеты

/// </summary>

private ArrayList \_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 ArrayList Exams

{

get => \_exams;

set => \_exams = value;

}

/// <summary>

/// Тесты

/// </summary>

public ArrayList 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 рождения

/// </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.Date = value.Date;

}

}

/// <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.Date)

{

\_education = education;

\_groupNumber = groupNumber;

}

/// <summary>

/// Конструктор без параметров

/// </summary>

public Student() : base()

{

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

\_education = EducationEnum.Вachelor;

}

/// <summary>

/// Добавление экзаменов

/// </summary>

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

public void AddExams(ArrayList exams)

{

\_exams = exams;

}

/// <summary>

/// Добавление тестов

/// </summary>

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

public void AddTests(ArrayList tests)

{

\_tests = tests;

}

/// <summary>

/// Получение строки со всеми полями класса

/// </summary>

/// <returns></returns>

public override string ToString()

{

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

{\_groupNumber}\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}\n";

}

/// <summary>

/// Получение полной копии

/// </summary>

/// <returns></returns>

public override object DeepCopy()

{

var newStudent = new Student((Person)base.DeepCopy(), Education,

GroupNumber);

var newExams = new ArrayList();

var newTests = new ArrayList();

foreach(var item in Exams)

{

newExams.Add(((Exam)item).DeepCopy());

}

foreach(var item in Tests)

{

newTests.Add(((Test)item).DeepCopy());

}

newStudent.AddExams(newExams);

newStudent.AddTests(newTests);

return newStudent;

}

/// <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;

}

}

}

1. Создаем новую версию класса Exam

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 Date { get; set; }

public Exam()

{

Subject = StringExtension.GetRandom(7);

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

Date = DateTime.UtcNow;

}

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

{

Subject = subject;

Grade = grade;

Date = date;

}

/// <summary>

/// Получение строки со всеми полями

/// </summary>

/// <returns></returns>

public override string ToString()

{

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

}

/// <summary>

/// Полуение полнй копии

/// </summary>

/// <returns></returns>

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

public object DeepCopy()

{

return new Exam(Subject, Grade, Date);

}

}

1. Определение интерфейса IdateAndCopy

public interface IDateAndCopy

{

/// <summary>

/// Копирование

/// </summary>

/// <returns></returns>

object DeepCopy();

/// <summary>

/// Дата

/// </summary>

DateTime Date { get; set; }

}

1. У студентов кроме экзаменов могут быть зачеты, для этого определяем класс Test

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);

}

}

1. Перечисление типов образования оставляем с первой лабораторной работы.

public enum EducationEnum

{

/// <summary>

/// Специалист

/// </summary>

Specialist,

/// <summary>

/// Балаклавр

/// </summary>

Вachelor,

/// <summary>

/// Второе образование

/// </summary>

SecondEducation

}

Консольный вывод:

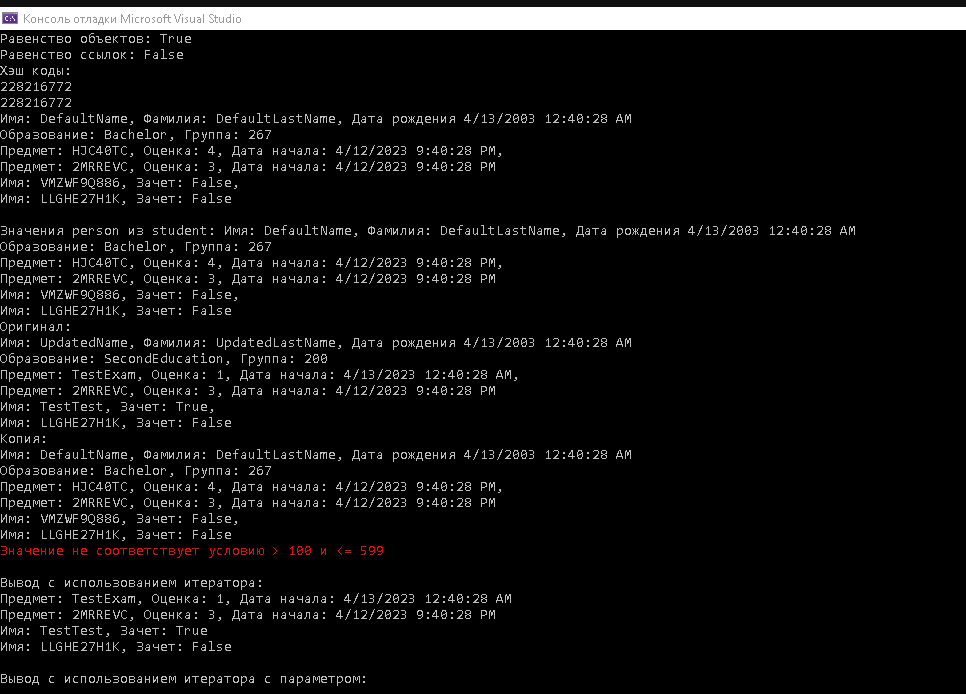


Рис. 1 Результат выполнения лабораторной работы №2