

**НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ ЧАСТНОЕУЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ«МОСКОВСКИЙ ФИНАНСОВО-ПРОМЫШЛЕННЫЙ УНИВЕРСИТЕТ“СИНЕРГИЯ”»**

|  |  |  |
| --- | --- | --- |
| **Факультет/Институт** |  | Информационных технологий |
|  |  | (наименование факультета/ Института) |
| **Направление/специальность** |  | Информационные системы и технологии |
| **подготовки:** |  | (код и наименование направления /специальности подготовки) |
| **Форма обучения:** |  | Очная |
|  |  | (очная, очно-заочная, заочная) |
|  |  |  |

**Отчет по практической работе №5**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **на тему** |  | Лабораторная работа №5 «Проектирование классов». | | |
|  |  | (наименование темы) | | |
|  |  |  | | |
| **по дисциплине** | | |  | **Разработка модулей программного обеспечения для компьютерных систем** |
|  | | |  | (наименование дисциплины) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Обучающийся** |  | Разумнов Данила Николавич |  | Не требуется |
|  |  | (ФИО) |  | (подпись) |
| **Группа** |  | Дкип-206Прог |  |  |
|  |  |  |  |  |

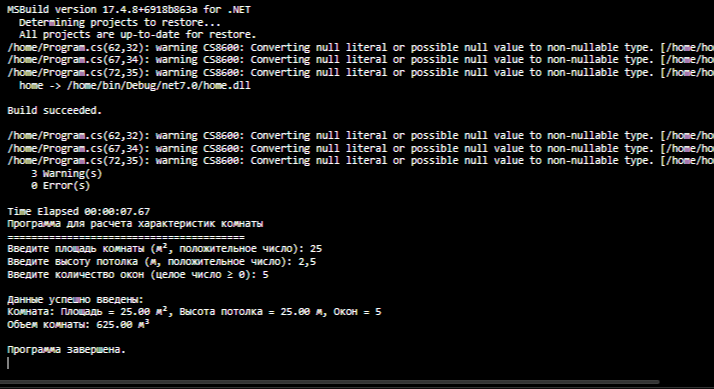
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Преподаватель** |  | **Сибирев Иван Валерьевич** |  |  |
|  |  | (ФИО) |  | (подпись) |

**Москва 2025 г.**

Лабораторная работа №5 «Проектирование классов».

Лабораторная работа 4. Простейшие классы

Вариант 16 Описать класс «комната», содержащий сведения о метраже, высоте потолков и количестве окон. Предусмотреть инициализацию с проверкой допустимости значений полей. В случае недопустимых значений полей выбрасываются исключения. Описать методы вычисления площади и объема комнаты и свойства для получения состояния объекта. Написать программу, демонстрирующую все разработанные элементы класса.



using System;

// Определение класса Room

public class Room

{

// Поля класса

private double area;

private double ceilingHeight;

private int windowCount;

// Конструктор с проверкой допустимости значений

public Room(double area, double ceilingHeight, int windowCount)

{

if (area <= 0)

throw new ArgumentException("Площадь комнаты должна быть положительной.");

if (ceilingHeight <= 0)

throw new ArgumentException("Высота потолка должна быть положительной.");

if (windowCount < 0)

throw new ArgumentException("Количество окон не может быть отрицательным.");

this.area = area;

this.ceilingHeight = ceilingHeight;

this.windowCount = windowCount;

}

// Метод вычисления площади

public double GetArea()

{

return area;

}

// Метод вычисления объема

public double CalculateVolume()

{

return area \* ceilingHeight;

}

// Свойства для получения состояния объекта

public double Area => area;

public double CeilingHeight => ceilingHeight;

public int WindowCount => windowCount;

// Метод для удобного вывода информации о комнате

public override string ToString()

{

return $"Комната: Площадь = {Area:F2} м², Высота потолка = {CeilingHeight:F2} м, Окон = {WindowCount}";

}

}

// Основная программа

class Program

{

static void Main()

{

Console.WriteLine("Программа для расчета характеристик комнаты");

Console.WriteLine("========================================");

try

{

// Ввод площади комнаты

Console.Write("Введите площадь комнаты (м², положительное число): ");

string areaInput = Console.ReadLine();

double area = Convert.ToDouble(areaInput);

// Ввод высоты потолка

Console.Write("Введите высоту потолка (м, положительное число): ");

string heightInput = Console.ReadLine();

double height = Convert.ToDouble(heightInput);

// Ввод количества окон

Console.Write("Введите количество окон (целое число ≥ 0): ");

string windowsInput = Console.ReadLine();

int windows = Convert.ToInt32(windowsInput);

// Создание объекта Room

Room room = new Room(area, height, windows);

// Вывод информации

Console.WriteLine("\nДанные успешно введены:");

Console.WriteLine(room.ToString());

Console.WriteLine($"Объем комнаты: {room.CalculateVolume():F2} м³");

}

catch (FormatException)

{

Console.WriteLine("\nОшибка: Введите корректное числовое значение.");

}

catch (OverflowException)

{

Console.WriteLine("\nОшибка: Значение слишком большое или маленькое.");

}

catch (ArgumentException ex)

{

Console.WriteLine($"\nОшибка: {ex.Message}");

}

catch (Exception ex)

{

Console.WriteLine($"\nПроизошла непредвиденная ошибка: {ex.Message}");

}

Console.WriteLine("\nПрограмма завершена.");

}

}

Лабораторная работа 8. Классы и операции Теоретический материал: глава 7. Каждый разрабатываемый класс должен, как правило, содержать следующие элементы: скрытые поля, конструкторы с параметрами и без параметров, методы; свойства, индексаторы; перегруженные операции. Функциональные элементы класса должны обеспечивать непротиворечивый, полный, минимальный и удобный интерфейс класса. При возникновении ошибок должны выбрасываться исключения. В программе должна выполняться проверка всех разработанных элементов класса.

Вариант 16 Описать класс «поезд», содержащий следующие закрытые поля: название пункта назначения; номер поезда (может содержать буквы и цифры); время отправления. Предусмотреть свойства для получения состояния объекта. Описать класс «вокзал», содержащий закрытый массив поездов. Обеспечить следующие возможности: вывод информации о поезде по номеру с помощью индекса; вывод информации о поездах, отправляющихся после введенного с клавиатуры времени; перегруженную операцию сравнения, выполняющую сравнение времени отправления двух поездов; вывод информации о поездах, отправляющихся в заданный пункт назначения. Информация должна быть отсортирована по времени отправления. Написать программу, демонстрирующую все разработанные элементы классов.



using System;

using System.Collections.Generic;

// Класс "Поезд"

public class Train

{

// Закрытые поля

private string destination;

private string trainNumber;

private TimeSpan departureTime;

// Свойства для получения состояния объекта

public string Destination => destination;

public string TrainNumber => trainNumber;

public TimeSpan DepartureTime => departureTime;

// Конструктор с проверкой корректности данных

public Train(string destination, string trainNumber, string departureTimeString)

{

if (string.IsNullOrWhiteSpace(destination))

throw new ArgumentException("Пункт назначения не может быть пустым.");

if (string.IsNullOrWhiteSpace(trainNumber))

throw new ArgumentException("Номер поезда не может быть пустым.");

if (!TimeSpan.TryParse(departureTimeString, out departureTime))

throw new ArgumentException("Некорректный формат времени отправления (ожидается HH:mm).");

}

// Перегрузка операторов сравнения

public static bool operator >(Train t1, Train t2)

{

return t1.DepartureTime > t2.DepartureTime;

}

public static bool operator <(Train t1, Train t2)

{

return t1.DepartureTime < t2.DepartureTime;

}

public static bool operator ==(Train t1, Train t2)

{

return t1?.DepartureTime == t2?.DepartureTime;

}

public static bool operator !=(Train t1, Train t2)

{

return !(t1 == t2);

}

public override string ToString()

{

return $"Поезд {TrainNumber} до {Destination}, отправление в {DepartureTime:hh\\:mm tt}";

}

}

// Класс "Вокзал"

public class Station

{

// Закрытый массив поездов

private Train[] trains;

// Конструктор

public Station(Train[] trains)

{

this.trains = trains;

}

// Индексатор для доступа к поездам по индексу

public Train this[int index]

{

get

{

if (index < 0 || index >= trains.Length)

throw new IndexOutOfRangeException("Индекс вне диапазона.");

return trains[index];

}

}

// Метод для вывода информации о поезде по номеру

public void PrintTrainByNumber(string trainNumber)

{

foreach (var train in trains)

{

if (train.TrainNumber == trainNumber)

{

Console.WriteLine(train);

return;

}

}

Console.WriteLine("Поезд с таким номером не найден.");

}

// Метод для вывода поездов, отправляющихся после заданного времени

public void PrintTrainsAfterTime(TimeSpan time)

{

Console.WriteLine($"\nПоезда, отправляющиеся после {time:hh\\:mm tt}:");

foreach (var train in trains)

{

if (train.DepartureTime > time)

{

Console.WriteLine(train);

}

}

}

// Метод для вывода поездов в заданный пункт назначения, отсортированных по времени

public void PrintTrainsToDestination(string destination)

{

List<Train> filteredTrains = new List<Train>();

foreach (var train in trains)

{

if (train.Destination.Equals(destination, StringComparison.OrdinalIgnoreCase))

{

filteredTrains.Add(train);

}

}

if (filteredTrains.Count == 0)

{

Console.WriteLine($"Постов в пункт {destination} не найдено.");

return;

}

// Сортировка по времени отправления

filteredTrains.Sort((t1, t2) => t1.DepartureTime.CompareTo(t2.DepartureTime));

Console.WriteLine($"\nПоезда в пункт {destination}, отсортированные по времени:");

foreach (var train in filteredTrains)

{

Console.WriteLine(train);

}

}

// Метод для сравнения двух поездов по времени отправления

public void CompareTrains(int index1, int index2)

{

Train train1 = this[index1];

Train train2 = this[index2];

if (train1 > train2)

{

Console.WriteLine($"Поезд {train1.TrainNumber} отправляется позже, чем {train2.TrainNumber}.");

}

else if (train1 < train2)

{

Console.WriteLine($"Поезд {train1.TrainNumber} отправляется раньше, чем {train2.TrainNumber}.");

}

else

{

Console.WriteLine($"Поезда {train1.TrainNumber} и {train2.TrainNumber} отправляются одновременно.");

}

}

}

// Основная программа

class Program

{

static void Main()

{

// Создание массива поездов

Train[] trains = new Train[]

{

new Train("Москва", "R123A", "08:00"),

new Train("Санкт-Петербург", "S456B", "10:30"),

new Train("Москва", "R789C", "12:45"),

new Train("Новосибирск", "N012D", "09:15"),

new Train("Москва", "R345E", "14:00")

};

// Создание вокзала

Station station = new Station(trains);

// Демонстрация работы с индексатором

Console.WriteLine("Поезд по индексу 2:");

Console.WriteLine(station[2]);

// Демонстрация поиска поезда по номеру

Console.WriteLine("\nПоиск поезда по номеру:");

station.PrintTrainByNumber("S456B");

// Демонстрация поиска поездов, отправляющихся после заданного времени

Console.WriteLine("\nВвод времени отправления для поиска (в формате HH:mm):");

string inputTime = Console.ReadLine();

if (TimeSpan.TryParse(inputTime, out TimeSpan time))

{

station.PrintTrainsAfterTime(time);

}

else

{

Console.WriteLine("Некорректный формат времени.");

}

// Демонстрация сравнения поездов

Console.WriteLine("\nСравнение поездов по индексам 0 и 3:");

station.CompareTrains(0, 3);

// Демонстрация поиска поездов в заданный пункт

Console.WriteLine("\nВвод пункта назначения для поиска:");

string destination = Console.ReadLine();

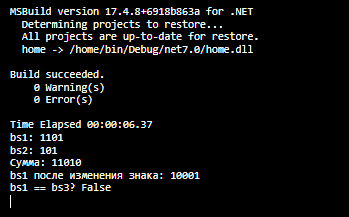
station.PrintTrainsToDestination(destination);

}

}

Лабораторная работа 9. Наследование Теоретический материал; глава 8, Теоретический материал: глана 8. В программах требуется описать базовый класс (возможно, абстрактный), в котором с помощью виртуальных или абстрактных методов и свойсти задается интерфейс для производных классов. Целью лабораторной работы является максимальное использование паследования, даже если для конкретной задачи оно не дает выигрыша в объеме программы. Во всех классах следует переопределить метод Equals, чтобы обеспечить сравнение значений, а не ссылок. Функция Маin должна содержать массив из элементов базового класса, занолнен- ный ссылками на производные классы. В этой функции должно демонстрироваться использование всех разработанных элементов классов.

Вариант 5 Описать базовый класс Строка в соответствии с вариантом 3. Описать производный от Строка класс Битовая строка. Строки данного класса могут содержать только символы '0' или '1'. Если в составе инициализирующей строки будут встречены любые символы, отличные от допустимых, класс Битовая строка принимает пулевое значение. Содержимое данных строк рассматривается как двоичное число. Отрицательные числа хранятся в дополнительном коде. Для класса Битовая строка определить следующие методы: конструктор, принимающий в качестве параметра строковый литерал; деструктор; изменение знака на противоположный (перевод числа в дополнительный код); присваивание; вычисление арифметической суммы строк; проверка на равенство. В случае необходимости более короткая битовая строка расширяется влево знаковым разрядом.



using System;

// Базовый класс "Строка"

public class StringBase

{

protected string value;

public StringBase(string val)

{

value = val;

}

public override string ToString()

{

return value;

}

// Деструктор (финализатор)

~StringBase()

{

Console.WriteLine("Объект StringBase удален.");

}

}

// Производный класс "Битовая строка"

public class BitString : StringBase

{

public BitString(string val) : base(val)

{

if (!IsBinary(val))

{

value = ""; // Принимаем пустое значение

}

}

// Проверка, является ли строка битовой

private bool IsBinary(string s)

{

foreach (char c in s)

{

if (c != '0' && c != '1')

return false;

}

return true;

}

// Метод присваивания

public void Assign(string val)

{

if (IsBinary(val))

{

value = val;

}

else

{

value = "";

}

}

// Метод изменения знака (дополнительный код)

public void Negate()

{

if (string.IsNullOrEmpty(value))

return;

// Инвертируем биты

char[] inverted = new char[value.Length];

for (int i = 0; i < value.Length; i++)

{

inverted[i] = (value[i] == '0') ? '1' : '0';

}

// Прибавляем 1

string twosComplement = AddBinary(new string(inverted), "1");

value = twosComplement;

}

// Метод сложения двух битовых строк

public static BitString Add(BitString a, BitString b)

{

if (a.value == null || b.value == null)

return new BitString("");

// Выравниваем длины строк

string aStr = a.value.PadLeft(b.value.Length, a.value[0]);

string bStr = b.value.PadLeft(a.value.Length, b.value[0]);

// Складываем

string result = AddBinary(aStr, bStr);

return new BitString(result);

}

// Вспомогательный метод сложения двух битовых строк

private static string AddBinary(string a, string b)

{

int length = Math.Max(a.Length, b.Length);

a = a.PadLeft(length, a[0]);

b = b.PadLeft(length, b[0]);

char[] result = new char[length + 1];

int carry = 0;

for (int i = length - 1; i >= 0; i--)

{

int sum = (a[i] - '0') + (b[i] - '0') + carry;

result[i + 1] = (sum % 2) == 0 ? '0' : '1';

carry = sum / 2;

}

result[0] = (carry % 2) == 0 ? '0' : '1';

return new string(result);

}

// Проверка на равенство

public bool Equals(BitString other)

{

if (string.IsNullOrEmpty(value) || string.IsNullOrEmpty(other.value))

return false;

// Выравниваем длины

string thisStr = value.PadLeft(other.value.Length, value[0]);

string otherStr = other.value.PadLeft(value.Length, other.value[0]);

return thisStr == otherStr;

}

// Деструктор

~BitString()

{

Console.WriteLine("Объект BitString удален.");

}

}

class Program

{

static void Main()

{

// Тестирование

BitString bs1 = new BitString("1101"); // -3 (дополнительный код 4 бита)

BitString bs2 = new BitString("101"); // 5 (3 бита)

Console.WriteLine("bs1: " + bs1);

Console.WriteLine("bs2: " + bs2);

// Сложение

BitString sum = BitString.Add(bs1, bs2);

Console.WriteLine("Сумма: " + sum);

// Изменение знака

bs1.Negate();

Console.WriteLine("bs1 после изменения знака: " + bs1);

// Проверка на равенство

BitString bs3 = new BitString("0011");

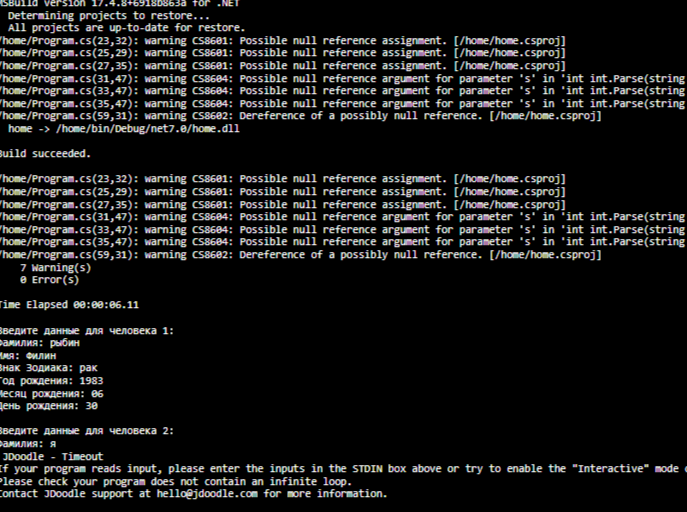
Console.WriteLine("bs1 == bs3? " + bs1.Equals(bs3));

}

}

Лабораторная работа 10. Структуры Теоретический материал: глава 9.

Вариант 16 Описать структуру с именем ZNAK, содержащую следующие поля: фамилия, имя; знак Зодиака; дата рождения (массив из трех чисел). Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в массив, состоящий из восьми элементов типа ZNAK (записи должны быть упорядочены по дате рождения); вывод на экран информации о людях, родившихся под знаком, название которого введено с клавиатуры (если таких нет, вывести соответствующее сообщение).



using System;

struct ZNAK

{

public string Surname;

public string Name;

public string ZodiacSign;

public int[] BirthDate; // [год, месяц, день]

}

class Program

{

static void Main()

{

const int COUNT = 8;

ZNAK[] znaki = new ZNAK[COUNT];

// Ввод данных

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

{

Console.WriteLine($"\nВведите данные для человека {i + 1}:");

Console.Write("Фамилия: ");

znaki[i].Surname = Console.ReadLine();

Console.Write("Имя: ");

znaki[i].Name = Console.ReadLine();

Console.Write("Знак Зодиака: ");

znaki[i].ZodiacSign = Console.ReadLine();

znaki[i].BirthDate = new int[3];

Console.Write("Год рождения: ");

znaki[i].BirthDate[0] = int.Parse(Console.ReadLine());

Console.Write("Месяц рождения: ");

znaki[i].BirthDate[1] = int.Parse(Console.ReadLine());

Console.Write("День рождения: ");

znaki[i].BirthDate[2] = int.Parse(Console.ReadLine());

}

// Сортировка по дате рождения

Array.Sort(znaki, (a, b) =>

{

for (int i = 0; i < 3; i++)

{

int cmp = a.BirthDate[i].CompareTo(b.BirthDate[i]);

if (cmp != 0)

return cmp;

}

return 0;

});

// Вывод отсортированного массива

Console.WriteLine("\nОтсортированные данные:");

foreach (var znak in znaki)

{

Console.WriteLine($"{znak.Surname} {znak.Name}, {znak.ZodiacSign}, {znak.BirthDate[0]}-{znak.BirthDate[1]}-{znak.BirthDate[2]}");

}

// Поиск по знаку Зодиака

Console.Write("\nВведите знак Зодиака для поиска: ");

string targetZodiac = Console.ReadLine().Trim().ToLower();

bool found = false;

Console.WriteLine($"\nЛюди с знаком {targetZodiac}:");

foreach (var znak in znaki)

{

if (znak.ZodiacSign.ToLower() == targetZodiac)

{

Console.WriteLine($"{znak.Surname} {znak.Name}, {znak.BirthDate[0]}-{znak.BirthDate[1]}-{znak.BirthDate[2]}");

found = true;

}

}

if (!found)

{

Console.WriteLine("Нет людей с этим знаком Зодиака.");

}

}

}