Program.cs

using AccommodationClass;

using ClientClass;

using ConsoleExtension;

using HotelDataNameSpace;

using HotelRoomClass;

using System;

using System.Linq;

namespace Program

{

internal class Program

{

static Accommodation[] Accommodations;

static Client[] Clients;

static HotelRoom[] HotelRooms;

static void Main() {

Console.BackgroundColor = ConsoleColor.White;

Console.ForegroundColor = ConsoleColor.Black;

Console.Title = "Гостиница - финансы";

Console.Clear();

// загрузка данных

Accommodations = Accommodation.Load("accommodations.txt");

Clients = Client.Load("clients.txt");

HotelRooms = HotelRoom.Load("hotelrooms.txt");

// название программы

Console.WriteLine("HotelFinances");

Console.WriteLine("Консольное приложение отслеживания финансов гостиницы");

Console.WriteLine();

// главный цикл программы

while (true)

{

Console.WriteLine("Главная");

Console.WriteLine();

Console.WriteLine("Клиенты");

Console.WriteLine("1) Добавить");

Console.WriteLine("2) Изменить");

Console.WriteLine("3) Список");

Console.WriteLine("4) Поиск");

Console.WriteLine("5) Удалить");

Console.WriteLine();

Console.WriteLine("Номера");

Console.WriteLine("6) Добавить");

Console.WriteLine("7) Изменить");

Console.WriteLine("8) Список");

Console.WriteLine("9) Поиск");

Console.WriteLine("10) Удалить");

Console.WriteLine();

Console.WriteLine("Поселение");

Console.WriteLine("11) Добавить");

Console.WriteLine("12) Изменить");

Console.WriteLine("13) Список");

Console.WriteLine("14) Поиск");

Console.WriteLine("15) Удалить");

Console.WriteLine();

Console.WriteLine("16) Отчёт о выручке");

Console.WriteLine();

Console.WriteLine("17) Выход из приложения");

Console.WriteLine();

// выбор действия пользователем

int userInput = CE.InputIntAvailable(

"Введите номер пункта меню: ",

Enumerable.Range(1, 17).ToArray(),

"Введите номер доступного действия"

);

Console.Clear();

switch (userInput)

{

case 1: Client.Add(ref Clients); break;

case 2: Client.Edit(ref Clients); break;

case 3: Client.View(ref Clients); break;

case 4: Client.Search(ref Clients); break;

case 5: Client.Delete(ref Clients); break;

case 6: HotelRoom.Add(ref HotelRooms); break;

case 7: HotelRoom.Edit(ref HotelRooms); break;

case 8: HotelRoom.View(ref HotelRooms); break;

case 9: HotelRoom.Search(ref HotelRooms); break;

case 10: HotelRoom.Delete(ref HotelRooms); break;

case 11: Accommodation.Add(ref Accommodations, ref HotelRooms); break;

case 12: Accommodation.Edit(ref Accommodations, ref HotelRooms); break;

case 13: Accommodation.View(ref Accommodations); break;

case 14: Accommodation.Search(ref Accommodations); break;

case 15: Accommodation.Delete(ref Accommodations); break;

case 16: CalculateRevenue(ref Accommodations, ref HotelRooms); break;

case 17:

HotelData.Save(Accommodations, "Accommodations.txt");

HotelData.Save(Clients, "Clients.txt");

HotelData.Save(HotelRooms, "HotelRooms.txt");

Environment.Exit(0);

break;

}

}

}

public static void CalculateRevenue(ref Accommodation[] accommodations, ref HotelRoom[] hotelRooms)

{

DateTime currentDateTime = CE.InputDateTime("Введите текущую дату: ");

decimal totalRevenue = accommodations

.Where(a => a.CheckOutDate <= currentDateTime)

.Sum(a => a.Revenue);

Console.WriteLine($"Общая выручка со всех номеров в период до {currentDateTime.ToString("dd.MM.yyyy")}: {totalRevenue}");

CE.WaitForEsc();

}

}

}

HotelData.cs

using ClientClass;

using ConsoleExtension;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

namespace HotelDataNameSpace

{

public static class HotelData

{

public static T[] Load<T>(string path) where T : class, IHotelDataItem

{

// создание файла в случае его отсутствия

if (!File.Exists(path))

{

// создаем пустой файл

using (FileStream fs = File.Create(path))

{

}

}

// получение текстовых данных из файла

string textData;

using (StreamReader sr = new StreamReader(path))

{

textData = sr.ReadToEnd();

}

// преобразование текста в строки параметров

string[] textDataItems = textData.Split(

new string[] { "\n\n" },

StringSplitOptions.None

);

// Создание массива для хранения объектов класса

T[] objects = new T[textDataItems.Where(tdi => !(new string[] { "\n", "" }).Contains(tdi.Trim())).ToArray().Length];

// Преобразование строк параметров в массив строчных параметров и создание объектов

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

{

string[] stringParams = textDataItems[i].Split(

new string[] { "\n" },

StringSplitOptions.RemoveEmptyEntries

);

if (stringParams.Length == 0)

{

Console.WriteLine();

return objects.Cast<T>().ToArray();

}

Console.WriteLine("lol");

// Создание экземпляра класса с помощью рефлексии

objects[i] = (T)Activator.CreateInstance(

typeof(T),

new object[] { stringParams }

);

}

return objects;

}

public static T[] Add<T>(IHotelDataItem[] array, string title, string messageResult, Dictionary<string, object> params\_) where T : IHotelDataItem, new()

{

Console.WriteLine(title);

Console.WriteLine();

List<T> list = array.Cast<T>().ToList();

// Создание экземпляра класса с помощью рефлексии

T hotelDataItem = (T)Activator.CreateInstance(

typeof(T),

new T().Input(params\_)

);

if (hotelDataItem == null)

{

CE.WaitForEsc();

return array.Cast<T>().ToArray();

}

list.Add(hotelDataItem);

list.Sort();

T[] newArray = list.Cast<T>().ToArray();

Console.WriteLine();

Console.WriteLine(messageResult);

CE.WaitForEsc();

return newArray;

}

public static IHotelDataItem[] Search(IHotelDataItem[] array, string searchTitle = "")

{

string searchString = CE.InputString($"{searchTitle}Поиск всем параметрам: ");

searchString = searchString.ToLower();

IHotelDataItem[] arraySearchResult = array.Where(

item => item.ToSearchableString().ToLower().Contains(searchString)

).ToArray();

if (array.Length == 0)

{

Console.WriteLine("Ничего не найдено");

}

else

{

Console.Clear();

View(arraySearchResult);

}

return arraySearchResult;

}

public static IHotelDataItem SelectiveSearch(IHotelDataItem[] array, string selectionTitle = "")

{

IHotelDataItem[] selectionResult = array;

do

{

selectionResult = Search(selectionResult);

Console.Clear();

View(selectionResult);

Console.WriteLine($"Количество результатов: {selectionResult.Length}");

}

while (selectionResult.Length > 1);

Console.Clear();

if (selectionResult.Length == 0)

{

Console.WriteLine("\nНичего не найдено");

return null;

}

Console.WriteLine("Результат поиска: \n");

IHotelDataItem searchResult = selectionResult[0];

Console.WriteLine(searchResult.ToString());

return searchResult;

}

public static void Save(IHotelDataItem[] array, string path)

{

// составление текстового представления

StringBuilder textData = new StringBuilder();

foreach (IHotelDataItem item in array)

{

textData.Append($"{string.Join("\n", item.ToData())}\n\n");

}

// запись в файл

using (StreamWriter sw = new StreamWriter(path))

{

sw.WriteLine(textData);

}

}

public static void View(IHotelDataItem[] array)

{

if (array.Length == 0)

{

Console.WriteLine("Список пуст");

return;

}

foreach (IHotelDataItem item in array)

{

Console.WriteLine(item.ToString());

}

}

public static void View(IHotelDataItem[] array, string message)

{

Console.Clear();

View(array);

Console.WriteLine();

Console.WriteLine(message);

Console.WriteLine($"Количество: {array.Length}");

CE.WaitForEsc();

}

public static IHotelDataItem[] Delete(IHotelDataItem[] array, string message)

{

Console.WriteLine(message);

Console.WriteLine();

IHotelDataItem searchResult = SelectiveSearch(array);

int userInput = CE.InputIntAvailable(

"Подтвердите удаление 1) да 2) нет: ",

Enumerable.Range(1, 2).ToArray()

);

Console.WriteLine();

switch (userInput)

{

case 1:

List<IHotelDataItem> list = array.ToList();

list.Remove(searchResult);

array = list.ToArray();

Console.WriteLine("Удаление завершено");

break;

case 2:

Console.WriteLine("Удаление отменено");

break;

}

CE.WaitForEsc();

return array;

}

public static IHotelDataItem[] Edit(IHotelDataItem[] array, string title, string searchTitle, string msgCompletion, Dictionary<string, object> params\_)

{

Console.WriteLine($"{title}\n");

IHotelDataItem hotelDataItem = SelectiveSearch(

array,

searchTitle

);

if (hotelDataItem == null)

{

CE.WaitForEsc();

return array;

}

Console.WriteLine("\nВведите новые данные\n");

object[] input = hotelDataItem.Input(params\_);

hotelDataItem.Edit(input);

Console.WriteLine($"\n{msgCompletion}\n");

CE.WaitForEsc();

return array;

}

}

public interface IHotelDataItem

{

void Edit(object[] input);

object[] Input(Dictionary<string, object> params\_);

string[] ToData();

string ToSearchableString();

string ToString();

}

}

Client.cs

using ConsoleExtension;

using HotelDataNameSpace;

using System;

using System.Collections.Generic;

using System.Linq;

namespace ClientClass

{

public class Client : IComparable<Client>, IHotelDataItem

{

// СВОЙСТВА

public string Comment { get; set; }

public string Name { get; set; }

public string PassportDetails { get; set; }

public string Patronymic { get; set; }

public string Surname { get; set; }

// МЕТОДЫ

public static Client[] Load(string path)

{

return HotelData.Load<Client>(path);

}

// МЕТОДЫ: КОНСТРУКТОРЫ

public Client() { }

public Client(string surname, string name, string patronymic, string passportDetails, string comment)

{

Surname = surname;

Name = name;

Patronymic = patronymic;

PassportDetails = passportDetails;

Comment = comment;

}

public Client(string[] stringParams)

{

Surname = stringParams[0];

Name = stringParams[1];

Patronymic = stringParams[2];

PassportDetails = stringParams[3];

Comment = stringParams[4];

}

// МЕТОДЫ: ДЕЙСТВИЯ ПОЛЬЗОВАТЕЛЯ

public static void Add(ref Client[] clients)

{

clients = HotelData

.Add<Client>(

clients,

"Добавление клиента",

"Клиент добавлен",

new Dictionary<string, object> { { "Client", (object)clients } }

)

.Cast<Client>()

.ToArray();

}

public static void Delete(ref Client[] clients)

{

clients = HotelData

.Delete(

clients,

"Удаление клиента"

)

.Cast<Client>()

.ToArray();

}

public static void Edit(ref Client[] clients)

{

clients = HotelData

.Edit(

clients,

"Изменение данных клиента",

"Выбор клиента - ",

"Данные клиента изменены",

new Dictionary<string, object>

{

{ "clients", clients }

}

)

.Cast<Client>()

.ToArray();

}

public static void Search(ref Client[] clients)

{

HotelData.Search(

clients,

"Клиенты - "

);

CE.WaitForEsc();

}

public static void View(ref Client[] clients)

{

HotelData.View(clients, "Просмотр списка клиентов");

}

// МЕТОДЫ: РЕАЛИЗАЦИЯ ИНТЕРФЕЙСОВ

int IComparable<Client>.CompareTo(Client other)

{

return this.Surname.CompareTo(other.Surname);

}

void IHotelDataItem.Edit(object[] input)

{

Surname = (string)input[0];

Name = (string)input[1];

Patronymic = (string)input[2];

PassportDetails = (string)input[3];

Comment = (string)input[4];

}

object[] IHotelDataItem.Input(Dictionary<string, object> params\_)

{

return new object[] {

CE.InputString("Фамилия: "),

CE.InputString("Имя: "),

CE.InputString("Отчество: "),

CE.InputString("Паспортные данные: "),

CE.InputString("Комментарий: ")

};

}

string[] IHotelDataItem.ToData()

{

return new string[] { Surname, Name, Patronymic, PassportDetails, Comment };

}

string IHotelDataItem.ToSearchableString()

{

return $"{Surname}{Name}{Patronymic}{PassportDetails}{Comment}";

}

string IHotelDataItem.ToString()

{

string result = $"Клиент \n";

result += $"Фамилия: {Surname} \n";

result += $"Имя: {Name} \n";

result += $"Отчество: {Patronymic} \n";

result += $"Паспортные данные: {PassportDetails} \n";

result += $"Комментарий: {Comment} \n";

return result;

}

}

}

Accommodation.cs

using ClientClass;

using ConsoleExtension;

using HotelDataNameSpace;

using HotelRoomClass;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Xml.Linq;

namespace AccommodationClass

{

public class Accommodation : IComparable<Accommodation>, IHotelDataItem

{

// СВОЙСТВА

public string ClientInitials { get; set; }

public int HotelRoomNumber { get; set; }

public DateTime CheckInDate { get; set; }

public DateTime CheckOutDate { get; set; }

public string Note { get; set; }

public decimal HotelRoomPrice { get; set; }

public decimal Revenue

{

get

{

return (CheckOutDate - CheckInDate).Days \* HotelRoomPrice;

}

}

// МЕТОДЫ

public static Accommodation[] Load(string path)

{

return HotelData.Load<Accommodation>(path);

}

// МЕТОДЫ: КОНСТРУКТОРЫ

public Accommodation() { }

public Accommodation(string clientInitials, int hotelRoomNumber, DateTime checkInDate, DateTime checkOutDate, string note, decimal hotelRoomPrice)

{

ClientInitials = clientInitials;

HotelRoomNumber = hotelRoomNumber;

CheckInDate = checkInDate;

CheckOutDate = checkOutDate;

Note = note;

HotelRoomPrice = hotelRoomPrice;

}

public Accommodation(string[] stringParams)

{

ClientInitials = stringParams[0];

HotelRoomNumber = int.Parse(stringParams[1]);

CheckInDate = DateTime.Parse(stringParams[2]);

CheckOutDate = DateTime.Parse(stringParams[3]);

Note = stringParams[4];

HotelRoomPrice = int.Parse(stringParams[5]);

}

// МЕТОДЫ: ДЕЙСТВИЯ ПОЛЬЗОВАТЕЛЯ

public static void Add(ref Accommodation[] accommodations, ref HotelRoom[] hotelRooms)

{

// заголовок операции

accommodations = HotelData

.Add<Accommodation>(

accommodations,

"Заселение клиента",

"Клиент заселён в комнату",

new Dictionary<string, object> {

{ "accommodations", accommodations},

{ "hotelRooms", hotelRooms }

}

)

.ToArray<Accommodation>();

}

public static void Delete(ref Accommodation[] accommodations)

{

accommodations = HotelData

.Delete(

accommodations,

"Удаление записи о заселении"

)

.Cast<Accommodation>()

.ToArray();

}

public static void Edit(ref Accommodation[] accommodations, ref HotelRoom[] hotelRooms)

{

accommodations = HotelData.Edit(

accommodations,

"Изменение данных записи о заселении",

"Выбор записи о заселении - ",

"Данные записи о заселении изменены",

new Dictionary<string, object>

{

{ "accommodations", accommodations },

{ "hotelRooms", hotelRooms }

}

)

.Cast<Accommodation>()

.ToArray();

}

public static void Search(ref Accommodation[] accommodations)

{

HotelData.Search(

accommodations,

"Записи о заселениях - "

);

CE.WaitForEsc();

}

public static void View(ref Accommodation[] accommodations)

{

HotelData.View(

accommodations,

"Просмотр записей о заселениях"

);

}

// МЕТОДЫ: РЕАЛИЗАЦИЯ ИНТЕРФЕЙСОВ

int IComparable<Accommodation>.CompareTo(Accommodation other)

{

return this.CheckInDate.CompareTo(other.CheckInDate);

}

void IHotelDataItem.Edit(object[] input)

{

ClientInitials = (string)input[0];

HotelRoomNumber = (int)input[1];

CheckInDate = (DateTime)input[2];

CheckOutDate = (DateTime)input[3];

Note = (string)input[4];

HotelRoomPrice = (decimal)input[5];

}

object[] IHotelDataItem.Input(Dictionary<string, object> params\_)

{

HotelRoom[] hotelRooms = (HotelRoom[])params\_["hotelRooms"];

Accommodation[] accommodations = (Accommodation[])params\_["accommodations"];

// проверка наличия свободных номеров

HotelRoom[] freeHotelRooms = HotelRoom.GetFreeHotelRooms(hotelRooms, accommodations);

if (freeHotelRooms.Length == 0)

{

Console.WriteLine("Все номера заняты");

CE.WaitForEsc();

return null;

}

// проверка соответствия бюджета свободным номерам

decimal budget = CE.InputDecimalPositive("Желаемая цена за ночь: ");

HotelRoom[] suitablePriceHotelRooms = freeHotelRooms

.Where(x => x.Price <= budget)

.ToArray<HotelRoom>();

if (suitablePriceHotelRooms.Count() == 0)

{

Console.WriteLine("Свободных номеров, подходящих по цене за ночь, не найдено");

return null;

}

// проверка соответствия требований к уровню комфорта

string desiredComfortLevel = HotelRoom.InputComfortLevel("Желаемый уровень комфорта (Люкс, Полулюкс, Обычный): ");

HotelRoom[] suitableHotelRooms = suitablePriceHotelRooms

.Where(x => x.ComfortLevel == desiredComfortLevel)

.ToArray<HotelRoom>();

if (suitableHotelRooms.Count() == 0)

{

Console.WriteLine("Подходящих номеров не найдено");

return null;

}

// вывод результатов проверки

int[] suitableHotelRoomsNumbers = suitableHotelRooms

.Select(x => x.Number)

.ToArray();

Console.WriteLine($"Подходящие номера ({suitableHotelRoomsNumbers.Count()}): {string.Join(", ", suitableHotelRoomsNumbers)}");

// заполнение других данных

string clientInitials = CE.InputString("ФИО клиента: ");

int hotelRoomNumber = CE.InputIntPositive(

"Номер: ",

input =>

{

if (!suitableHotelRoomsNumbers.Contains(input))

{

Console.WriteLine($"Введите номер доступного помещения ({string.Join(", ", suitableHotelRoomsNumbers)})");

return false;

}

return true;

}

);

DateTime checkInDate = CE.InputDateTime(

"Дата заселения: ",

input => // проверка задних чисел

{

if (input.Subtract(DateTime.Now).Days <= 0)

{

Console.WriteLine("Нельзя ввести прошедшую дату");

return false;

}

return true;

}

);

DateTime checkOutDate = CE.InputDateTime(

"Дата выселения: ",

input => // проверка неположительного периода заселения (задние числа уже не могут быть)

{

if (input.Subtract(checkInDate).Days <= 0)

{

Console.WriteLine("Дата выселения не может быть в один день или раньше заселения");

return false;

}

return true;

}

);

string note = CE.InputString("Примечание: ");

// возврат параметров

return new object[]

{

clientInitials,

hotelRoomNumber,

checkInDate,

checkOutDate,

note,

hotelRooms.First(hr => hr.Number == hotelRoomNumber).Price

};

}

string[] IHotelDataItem.ToData()

{

return new string[] {

ClientInitials,

HotelRoomNumber.ToString(),

CheckInDate.ToString(),

CheckOutDate.ToString(),

Note,

HotelRoomPrice.ToString(),

};

}

string IHotelDataItem.ToSearchableString()

{

return $"{ClientInitials}{HotelRoomNumber}{CheckInDate}{CheckOutDate}{Note}";

}

string IHotelDataItem.ToString()

{

string result = $"Поселение \n";

result += $"Клиент: {ClientInitials} \n";

result += $"Номер: {HotelRoomNumber} \n";

result += $"Дата поселения: {CheckInDate} \n";

result += $"Дата освобождения: {CheckOutDate} \n";

result += $"Примечание: {Note} \n";

return result;

}

}

}

HotelRoom.cs

using AccommodationClass;

using ClientClass;

using ConsoleExtension;

using HotelDataNameSpace;

using System;

using System.Collections.Generic;

using System.Linq;

namespace HotelRoomClass

{

public class HotelRoom : IComparable<HotelRoom>, IHotelDataItem

{

// СВОЙСТВА

public int Number { get; set; }

public decimal Price { get; set; }

public int PersonsNumber { get; set; }

public static string[] availableComfortLevel { get; set; } = new string[] { "Люкс", "Полулюкс", "Обычный" };

private string comfortLevel;

public string ComfortLevel

{

get

{

return comfortLevel;

}

set

{

if (!availableComfortLevel.Contains(value))

{

throw new ArgumentException("Недопустимое значения атрибута комфорта");

}

comfortLevel = value;

}

}

// МЕТОДЫ

public static HotelRoom[] Load(string path)

{

return HotelData.Load<HotelRoom>(path);

}

public static HotelRoom[] GetFreeHotelRooms(HotelRoom[] hotelRooms, Accommodation[] accommodations)

{

List<HotelRoom> freeHotelRooms = new List<HotelRoom>();

foreach (HotelRoom hotelRoom in hotelRooms)

{

bool hotelRoomIsFree = hotelRoom.PersonsNumber > accommodations

.Where(a => a.HotelRoomNumber == hotelRoom.Number)

.Count();

if (hotelRoomIsFree)

{

freeHotelRooms.Add(hotelRoom);

}

}

if (freeHotelRooms.Count == 0)

{

Console.WriteLine("Свободные номера отсутствуют");

}

return freeHotelRooms.ToArray();

}

public static string InputComfortLevel(string message)

{

return CE.InputString(

message,

input =>

{

if (availableComfortLevel.Contains(input))

{

return true;

}

Console.WriteLine($"Введите доступное значение");

return false;

}

);

}

// МЕТОДЫ: КОНСТРУКТОРЫ

public HotelRoom() { }

public HotelRoom(int number, int personsNumber, string comfortLevel, decimal price)

{

Number = number;

PersonsNumber = personsNumber;

ComfortLevel = comfortLevel;

Price = price;

}

public HotelRoom(string[] stringParams)

{

Number = int.Parse(stringParams[0]);

PersonsNumber = int.Parse(stringParams[1]);

ComfortLevel = stringParams[2];

Price = decimal.Parse(stringParams[3]);

}

// МЕТОДЫ: ДЕЙСТВИЯ ПОЛЬЗОВАТЕЛЯ

public static void Add(ref HotelRoom[] hotelRooms)

{

hotelRooms = HotelData

.Add<HotelRoom>(

hotelRooms,

"Добавление номера",

"Данные о номере добавлены",

new Dictionary<string, object> { { "HotelRooms", hotelRooms } }

)

.Cast<HotelRoom>()

.ToArray();

}

public static void Delete(ref HotelRoom[] hotelRooms)

{

hotelRooms = HotelData

.Delete(

hotelRooms,

"Удаление номера"

)

.Cast<HotelRoom>()

.ToArray();

}

public static void Edit(ref HotelRoom[] hotelRooms)

{

hotelRooms = HotelData

.Edit(

hotelRooms,

"Изменение данных клиента",

"Выбор номера - ",

"Данные номера изменены",

new Dictionary<string, object> {

{ "hotelRooms", hotelRooms }

}

)

.Cast<HotelRoom>()

.ToArray(); ;

}

public static void Search(ref HotelRoom[] hotelRooms)

{

HotelDataNameSpace.HotelData.Search(

hotelRooms,

"Номера - "

);

CE.WaitForEsc();

}

public static void View(ref HotelRoom[] hotelRooms)

{

HotelDataNameSpace.HotelData.View(

hotelRooms,

"Просмотр списка номеров"

);

}

// МЕТОДЫ: РЕАЛИЗАЦИЯ ИНТЕРФЕЙСОВ

int IComparable<HotelRoom>.CompareTo(HotelRoom other)

{

return this.Number.CompareTo(other.Number);

}

void IHotelDataItem.Edit(object[] input)

{

Number = (int)input[0];

PersonsNumber = (int)input[1];

ComfortLevel = (string)input[2];

Price = (decimal)input[3];

}

object[] IHotelDataItem.Input(Dictionary<string, object> params\_)

{

HotelRoom[] hotelRooms = (HotelRoom[])params\_["hotelRooms"];

IEnumerable<int> existingRoomNumbers = hotelRooms.Select(hr => hr.Number);

return new object[]

{

CE.InputIntPositive(

"Номер(число): ",

input =>

{

if (input <= 0)

{

Console.WriteLine("Номер должен быть положительным");

return false;

}

if (existingRoomNumbers.Contains(input))

{

Console.WriteLine($"Номер уже добавлен. Существующие номера {string.Join(", ", existingRoomNumbers)}");

return false;

}

return true;

}

),

CE.InputIntPositive(

"Вместимость (кол-во человек): ",

input =>

{

if (input <= 0)

{

Console.WriteLine("Количество должно быть положительным");

return false;

}

return true;

}

),

InputComfortLevel("Комфортность (Люкс, Полулюкс, Обычный): "),

CE.InputDecimalPositive(

"Цена за ночь: ",

input =>

{

if (input <= 0)

{

Console.WriteLine("Цена должна быть положительной");

return false;

}

return true;

}

)

};

}

string[] IHotelDataItem.ToData()

{

return new string[] {

Number.ToString(),

PersonsNumber.ToString(),

ComfortLevel,

Price.ToString()

};

}

string IHotelDataItem.ToSearchableString()

{

return $"{Number}{Price}{PersonsNumber}{ComfortLevel}";

}

string IHotelDataItem.ToString()

{

string result = $"Номер в отеле \n";

result += $"Номер: {Number} \n";

result += $"Количество человек: {PersonsNumber} \n";

result += $"Комфортность: {ComfortLevel} \n";

result += $"Цена: {Price:C} \n";

return result;

}

}

}

ConsoleExtension.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Cryptography;

namespace ConsoleExtension

{

public static class CE

{

public static void WaitForEsc()

{

Console.WriteLine("\nEsc) Назад в меню");

while (!(Console.ReadKey(true).Key == ConsoleKey.Escape)) ;

Console.Clear();

}

public static int InputInt(string message)

{

int input = 0;

bool correctInput = false;

do

{

try

{

Console.Write(message);

input = int.Parse(Console.ReadLine());

correctInput = true;

}

catch

{

Console.WriteLine("Введите число");

}

}

while (!correctInput);

return input;

}

public static int InputIntPositive(string message)

{

int input;

do

{

input = InputInt(message);

if (input > 0)

{

break;

}

Console.WriteLine("Число должны быть положительным");

}

while (true);

return input;

}

public static int InputIntPositive(string message, Func<int, bool> checkInput)

{

int input = 0;

do

{

input = InputIntPositive(message);

if (checkInput(input)) {

break;

};

}

while (true);

return input;

}

public static int InputIntAvailable(string message, int[] availableInput)

{

int input = 0;

do

{

input = InputIntPositive(message);

if (availableInput.Contains(input))

{

break;

}

Console.WriteLine($"Введите допустимое значение ({string.Join(", ", availableInput)})");

}

while (true);

return input;

}

public static int InputIntAvailable(string message, int[] availableInput, string messageAvailable = "Введите допустимое значение")

{

int input = 0;

do

{

input = InputIntPositive(message);

if (availableInput.Contains(input))

{

break;

}

Console.WriteLine($"{messageAvailable} ({string.Join(", ", availableInput)})");

}

while (true);

return input;

}

public static decimal InputDecimal(string message)

{

decimal input = 0;

bool correctInput = false;

do

{

try

{

Console.Write(message);

input = decimal.Parse(Console.ReadLine());

correctInput = true;

}

catch

{

Console.WriteLine("Введите числовое значение");

}

}

while (!correctInput);

return input;

}

public static decimal InputDecimalPositive(string message)

{

decimal input = 0;

do

{

input = InputDecimal(message);

if (input > 0)

{

break;

}

Console.WriteLine("Введите положительное число");

}

while (true);

return input;

}

public static decimal InputDecimalPositive(string message, Func<decimal, bool> checkInput)

{

decimal input;

do

{

input = InputDecimalPositive(message);

if (checkInput(input))

{

break;

}

}

while (true);

return input;

}

public static DateTime InputDateTime(string message)

{

DateTime input = new DateTime();

bool correctInput = false;

do

{

try

{

Console.Write(message);

input = DateTime.ParseExact(Console.ReadLine(), "d", null);

correctInput = true;

}

catch

{

Console.WriteLine("Введите верный формат даты (дд.мм.гггг)");

}

}

while (!correctInput);

return input;

}

public static DateTime InputDateTime(string message, Func<DateTime, bool> checkInput)

{

DateTime input = new DateTime();

do

{

input = InputDateTime(message);

if (checkInput(input))

{

break;

}

}

while (true);

return input;

}

public static string InputString(string message)

{

string input;

do

{

Console.Write(message);

input = Console.ReadLine().Trim();

if (!(input.Length == 0))

{

break;

}

Console.WriteLine("Строка не может быть пустой");

}

while (true);

return input;

}

public static string InputString(string message, Func<string, bool> checkInput)

{

string input;

do

{

input = InputString(message);

if (checkInput(input))

{

break;

}

}

while (true);

return input;

}

}

}