**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

по дисциплине: «Объектно-ориентированное проектирование и   
программирование»

на тему: **«**Наследование классов**»**

Выполнил: студент гр. ИТИ-21

Мусафиров А.Ю.

Принял: преподаватель

Башаримов Ю. С.

Гомель 2023

**Цель работы**: изучить организацию наследования классов, узнать методы и механизмы наследования, разработать иерархию классов.

**Условие задания:**

1. Необходимо решить задачу, согласно варианта;

2. При создании классов руководствоваться Code Convention;

3. Весь код должен содержать элементы документирования;

4. При реализации иерархии классов использовать механизм наследования;

5. Разработать модульные тесты для верификации созданных классов;

6. Классы должен быть размещены в библиотеке классов;

7. Модульные тесты - в отдельном проекте;

**Задание:**

1. Разработать иерархию классов для хранения информации O приобретённых билетах на автобус, двигающийся по маршруту Гомель - Речица - Светлогорск - Жлобин - Бобруйск. Количество мест в автобусе ограничено и зависит от конкретного типа автобуса (Маз, Мерседес, Ман и т.д.)
2. Стоимость билета зависит от типа автобуса и расстояния, т.е. билет

Гомель-Речица будет дешевле, чем Речица - Жлобин (ч/з Светлогорск)

1. Обработать ситуацию, если данный билет не может быть продан, т.к. нет мест.
2. Учесть, что пассажиры могут ехать по части маршрута и на освободившиеся места могут быть проданы билеты.
3. Вывести информацию, сколько осталось свободных билетов по каждому из возможных маршрутов (10 маршрутов)

**Ход работы**

В проекте мы реализуем иерархию классов следующим путём: от класса *Bus* будут наследоваться такие классы как *MAZ, Mersedes* и *Man.* В первом классе будет хранится базовая информация о транспорте, а в остальных классах более подробная, к примеру, начальная стоимость поездки или количество мест на определённом виде автобуса.

На рисунке 1 изображен процесс создания записи для поездки по вводимому маршруту.

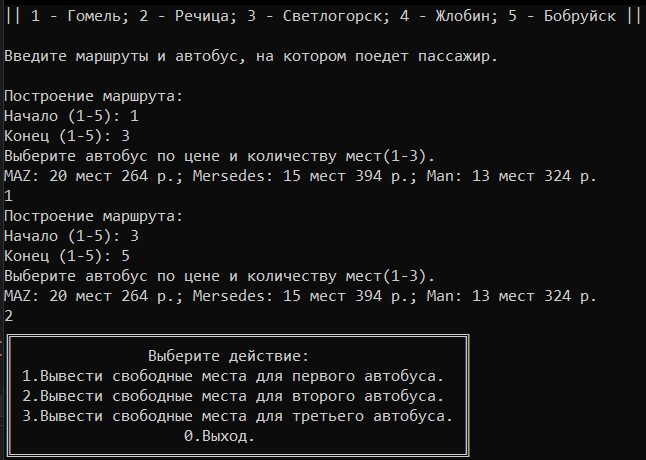


Рисунок 1 – Процесс создания записи.

Стоит отметить, что пункты назначения вводятся в виде цифр (1 - Гомель; 2 - Речица; 3 - Светлогорск; 4 - Жлобин; 5 - Бобруйск). Сделано это с целю упрощения вычисления «расстояния» маршрута. Путь находится путём вычитания от номера пункта высадки номера пункта посадки. Благодаря такому подходу можно с лёгкостью вычислить стоимость поездки только на основе данных о расстоянии. Найти данную стоимость можно по формуле (1):

|  |  |
| --- | --- |
|  | (1) |

где *x* – номер начального пункта;

*y* – номер конечного пункта;

*a−* произвольный коэффициент.

Полная же стоимость складывается из двух параметров: начальная стоимость поездки на определённом транспорте и нашей стоимости, найденной на основе данных о расстоянии. Нахождение полной стоимости представлено в формуле (2):

|  |  |
| --- | --- |
|  | (2) |

где *bus coast* – начальная стоимость поездки;

*coast* – стоимость поездки из расстояния.

Разницу в цене можно увидеть на рисунках 2 и 3.

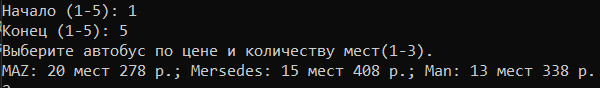


Рисунок 2 – Цены на автобусы на длинном маршруте.

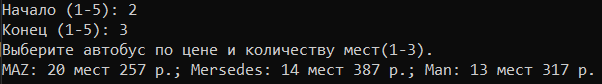


Рисунок 3 – Цены на автобусы на коротком маршруте.

На рисунке 4 изображён результат вывода записи в указанный день.

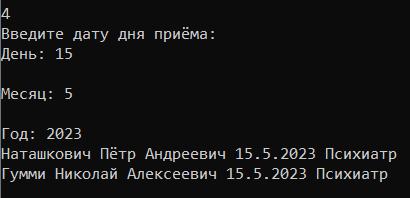


Рисунок 4 – Результат работы метода для вывода записи в указанный день.

На рисунке 5 изображена работа вывода количества пациента у доктора в определённый день.

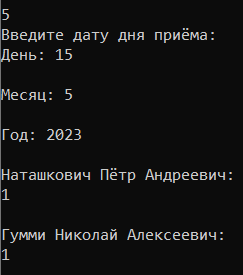


Рисунок 5 – Окно консоли при выводе количества пациентов у доктора в указанный день.

На рисунке 6 изображён результат вывода среднего количества пациента в день по специальности доктора.

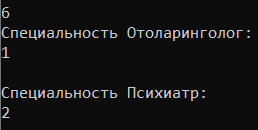


Рисунок 6 – Окно консоли при работе метода вывода среднего количества пациентов в день по специальности.

В приложении А и Б представлен код программы и код библиотеки.

На рисунке 7 уже изображён результат тестов, разработанных для проверки методов, созданных внутри класса для записей:

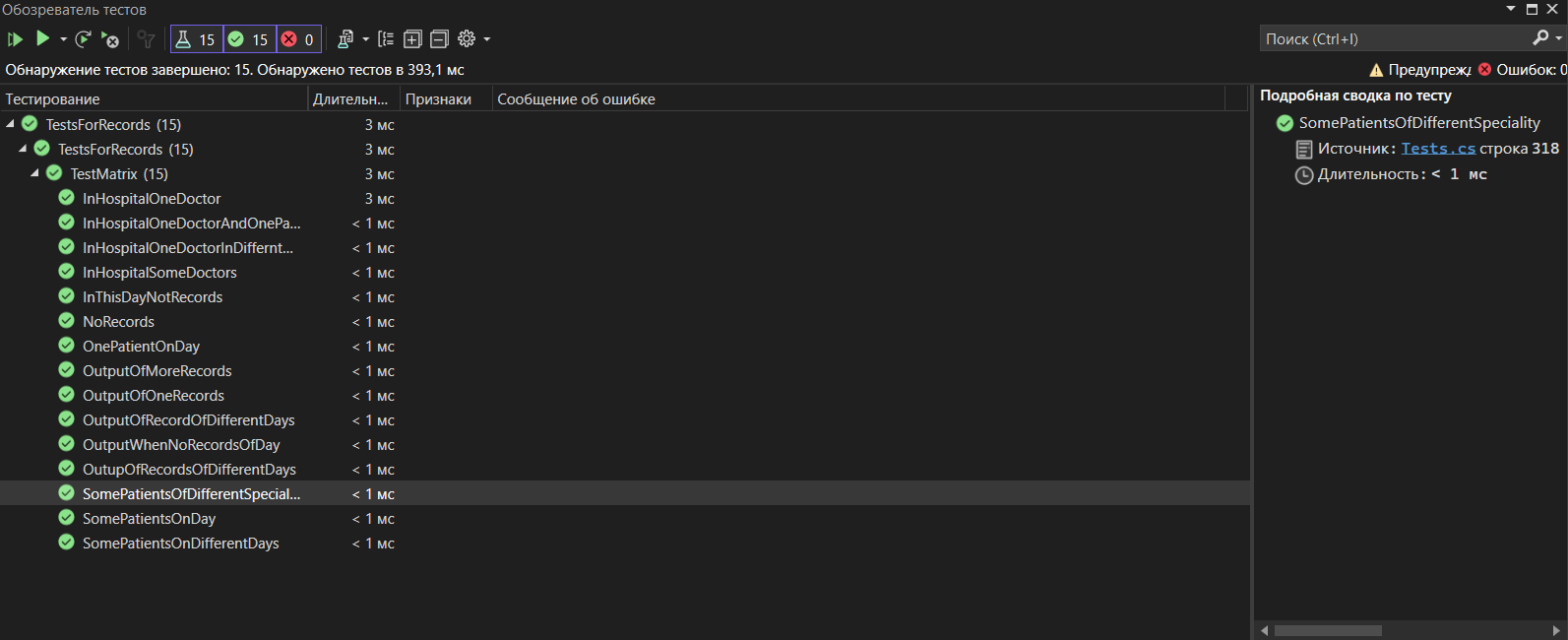


Рисунок 7 – *15* пройденных модульных тестов.

На рисунке 8 показан журнал *Git* с путями:

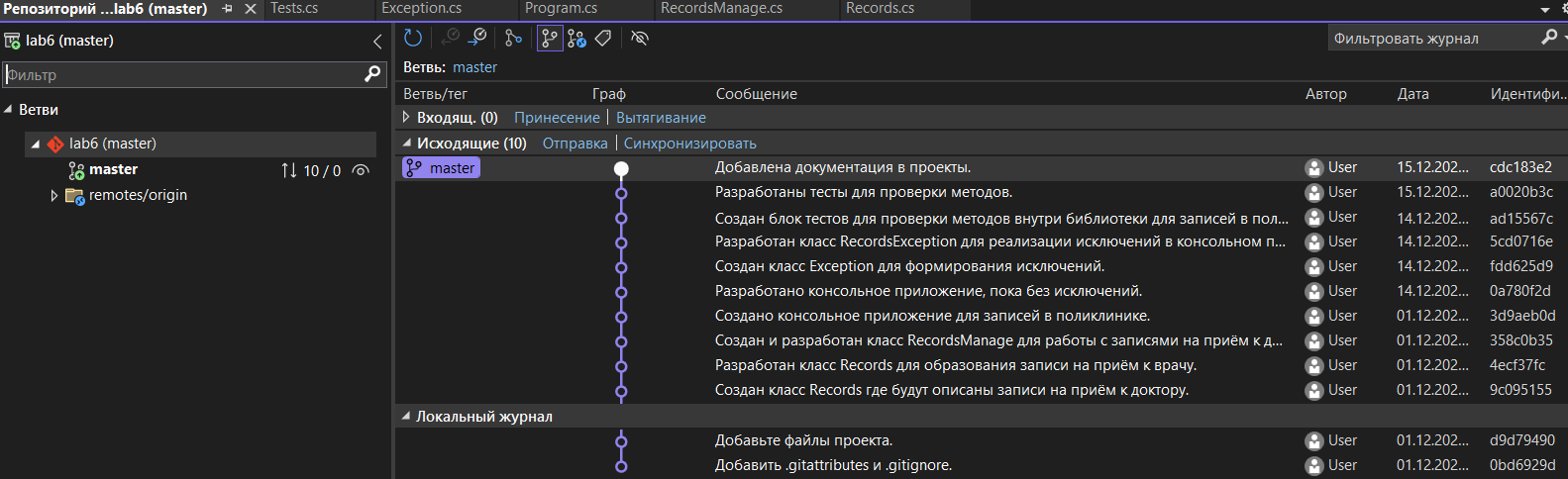


Рисунок 8 – Журнал *Git*.

Так как тесты все верны, то можно сделать вывод, что методы были разработаны правильно, а именно все действия, которые должны были выполнятся по условию.

В приложении В представлен код модульных тестов.

При выполнении данной работы были сделаны следующие вещи:

– создан классы для описания записи и списка записей;

– создана и подключена к основной части библиотека;

– разработаны методы для выполнения поставленных задач;

– организовано консольное меню пользователя;

– сформированы методы исключения для ввода данных;

– создано перечисления специальностей;

– выполнены модульные тесты;

**Вывод:** обучились организации обработки данных, научились предоставлять доступ к данным объекта, использовать перечисление, рассмотрели методы и механизмы инкапсуляции.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Текст программы**

**Program.cs:**

using System;

using LibraryForRecords;

namespace Hospital

{

class Program

{

static void Main(string[] args)

{

Console.Write("Введите количество записей: ");

int numberOfRecords = 0;

bool ValidateValue = true;

while (ValidateValue)

{

try

{

string number = Console.ReadLine();

RecordsException.ValidateInt(number, out numberOfRecords);

ValidateValue = false;

}

catch (RecordsException ex)

{

Console.WriteLine($"Ошибка ввода: {ex.Message}");

}

}

RecordsManage recordsManage = new RecordsManage(numberOfRecords);

string result = "";

bool exit = false;

while (!exit)

{

Console.WriteLine("\n╔═══════════════════════════════════════════════════════════════════╗");

Console.WriteLine("║ Выберите действие: ║");

Console.WriteLine("║ 1. Ввод записи на приём ║");

Console.WriteLine("║ 2. Редактирование записей ║");

Console.WriteLine("║ 3. Просмотр записей ║");

Console.WriteLine("║ 4. Вывод записей на определённый день ║");

Console.WriteLine("║ 5. Вывод количества пациентов по доктору в определённый день ║");

Console.WriteLine("║ 6. Подсчёт среднего количества пациентов в день по специальности ║");

Console.WriteLine("║ 0. Выход ║");

Console.WriteLine("╚═══════════════════════════════════════════════════════════════════╝\n");

int choice = 0;

ValidateValue = true;

while (ValidateValue)

{

try

{

string number = Console.ReadLine();

RecordsException.ValidateInt(number, out choice);

ValidateValue = false;

}

catch (RecordsException ex)

{

Console.WriteLine($"Ошибка ввода: {ex.Message}\n");

}

}

switch (choice)

{

case 1: // Ввод записей.

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

{

Console.WriteLine($"Введите данные для записи {i + 1}:");

Console.Write("ФИО доктора: ");

bool empty = true;

while (empty)

{

recordsManage.records[i].Initials = Console.ReadLine();

if (recordsManage.records[i].Initials != "")

{

empty = false;

}

else

{

Console.WriteLine("Строка пуста!");

}

}

Console.Write("Дата: ");

recordsManage.records[i].Time = Program.InputData();

Console.Write($"Специальность(1.Кардиолог;2.Отоларинголог;3.Психиатр;4.Ревматолог;5.Офтальмолог): ");

recordsManage.records[i].Speciality = 0;

ValidateValue = true;

while (ValidateValue)

{

try

{

string number = Console.ReadLine();

RecordsException.ValidateSpeciality(number, out recordsManage.records[i].Speciality);

ValidateValue = false;

}

catch (RecordsException ex)

{

Console.WriteLine($"Ошибка ввода: {ex.Message}\n");

}

}

Console.WriteLine();

}

break;

case 2: // Редактирование записи.

Console.Write("Введите номер записи для редактирования: ");

int recordNumber = 0;

ValidateValue = true;

while (ValidateValue)

{

try

{

string number = Console.ReadLine();

RecordsException.ValidateInt(number, out recordNumber);

recordNumber--;

ValidateValue = false;

}

catch (RecordsException ex)

{

Console.WriteLine($"Ошибка ввода: {ex.Message}\n");

}

}

if (recordNumber >= 0 && recordNumber < numberOfRecords)

{

Console.WriteLine($"----------- Редактирование данных для записи {recordNumber + 1} -----------");

Console.Write("\nФИО доктора: ");

bool empty = true;

while (empty)

{

recordsManage.records[recordNumber].Initials = Console.ReadLine();

if (recordsManage.records[recordNumber].Initials != "")

{

empty = false;

}

else

{

Console.WriteLine("Строка пуста!");

}

}

Console.Write("Дата: ");

recordsManage.records[recordNumber].Time = Program.InputData();

Console.Write($"Специальность(1.Кардиолог;2.Отоларинголог;3.Психиатр;4.Ревматолог;5.Офтальмолог): ");

recordsManage.records[recordNumber].Speciality = 0;

ValidateValue = true;

while (ValidateValue)

{

try

{

string number = Console.ReadLine();

RecordsException.ValidateSpeciality(number, out recordsManage.records[recordNumber].Speciality);

ValidateValue = false;

}

catch (RecordsException ex)

{

Console.WriteLine($"Ошибка ввода: {ex.Message}\n");

}

}

Console.WriteLine("----------- Данные успешно отредактированы -----------\n");

}

else

{

Console.WriteLine("Записи с таким номером не существует.\n");

}

break;

case 3: // Вывод записей.

Console.WriteLine("Записи в поликлинике:");

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

{

if (recordsManage.records[i].Initials != null)

{

Console.WriteLine("\n══════════════════════════════════════════════════════════════════════════════════════");

Console.WriteLine($"\n Запись {i + 1}");

Console.WriteLine($" ФИО доктора: {recordsManage.records[i].Initials}");

Console.WriteLine($" Дата: {recordsManage.records[i].Time}");

Console.WriteLine($" Специальность(1.Кардиолог;2.Отоларинголог;3.Психиатр;4.Ревматолог;5.Офтальмолог): {recordsManage.records[i].Speciality}");

Console.WriteLine("\n═══════════════════════════════════════════════════════════════════════════════════════\n");

}

}

break;

case 4: // Вывод записей на определённый день.

Console.Write("Введите дату дня приёма: ");

string time = Program.InputData();

result = RecordsManage.TimeRecords(time,recordsManage.records);

Console.WriteLine(result);

break;

case 5: // Вывод количества пациентов по доктору в определённый день.

Console.Write("Введите дату дня приёма: ");

time = Program.InputData();

Console.WriteLine();

result = RecordsManage.DoctorRecords(time,recordsManage.records);

Console.WriteLine(result);

break;

case 6: // Подсчёт среднего количества пациентов в день по специальности.

result = RecordsManage.AveragePatients(recordsManage.records);

Console.WriteLine(result);

break;

case 0:

exit = true;

break;

default:

Console.WriteLine("Некорректный выбор. Пожалуйста, выберите существующую опцию.");

break;

}

}

}

/// <summary>

/// Метод ввода даты.

/// </summary>

/// <returns></returns>

private static string InputData()

{

int day, month, year;

string time;

bool ValidateValue = true;

Console.Write("\nДень: ");

day = 0;

ValidateValue = true;

while (ValidateValue)

{

try

{

string number = Console.ReadLine();

RecordsException.ValidateDay(number, out day);

ValidateValue = false;

}

catch (RecordsException ex)

{

Console.WriteLine($"Ошибка ввода: {ex.Message}\n");

}

}

Console.Write("\nМесяц: ");

month = 0;

ValidateValue = true;

while (ValidateValue)

{

try

{

string number = Console.ReadLine();

RecordsException.ValidateMonth(number, out month);

ValidateValue = false;

}

catch (RecordsException ex)

{

Console.WriteLine($"Ошибка ввода: {ex.Message}\n");

}

}

Console.Write("\nГод: ");

year = 0;

ValidateValue = true;

while (ValidateValue)

{

try

{

string number = Console.ReadLine();

RecordsException.ValidateYear(number, out year);

ValidateValue = false;

}

catch (RecordsException ex)

{

Console.WriteLine($"Ошибка ввода: {ex.Message}\n");

}

}

time = day + "." + month + "." + year;

return time;

}

}

}

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Тексты библиотеки**

**Records.cs:**

namespace LibraryForRecords

{

/// <summary>

/// Класс о информации в записи.

/// </summary>

public class Records

{

public string Initials;

public string Time;

public int Speciality;

/// <summary>

/// Пустой конструктор.

/// </summary>

public Records()

{

this.Initials = "";

this.Time = "00.00.0000";

this.Speciality = 0;

}

/// <summary>

/// Конструктор с параметрами.

/// </summary>

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

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

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

public Records(string initials, string time, int speciality)

{

Initials = initials;

Time = time;

Speciality = speciality;

}

}

}

**RecordsManage.cs:**

namespace LibraryForRecords

{

/// <summary>

/// Нумерация по специальности.

/// </summary>

enum Speciality

{

Кардиолог = 1,

Отоларинголог = 2,

Психиатр = 3,

Ревматолог = 4,

Офтальмолог = 5,

}

/// <summary>

/// Класс с хранением записей.

/// </summary>

public class RecordsManage

{

/// <summary>

/// Поле записей.

/// </summary>

public Records[] records;

/// <summary>

/// Конструктор по количеству записей.

/// </summary>

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

public RecordsManage(int NumberOfRecords)

{

records = new Records[NumberOfRecords];

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

{

records[i] = new Records();

}

}

/// <summary>

/// Метод для вывода значений констант из enum по передаваемому индексу.

/// </summary>

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

/// <returns></returns>

private static string EnumerationExtract(int index)

{

// Получение всех значений перечисления в массив

Speciality[] specialitiesArray = (Speciality[])Enum.GetValues(typeof(Speciality));

string spec = "";

index--;

// Обращение к элементу по индексу

if (index >= 0 && index < specialitiesArray.Length)

{

spec = specialitiesArray[index].ToString();

}

return spec;

}

/// <summary>

/// Метод, для вывода записей в определённый день.

/// </summary>

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

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

/// <returns></returns>

public static string TimeRecords(string time, Records[] records)

{

int count = 0;

string timeRecords = "";

foreach (var record in records)

{

if (record.Time == time)

{

timeRecords = timeRecords + record.Initials + " " + record.Time + " " + EnumerationExtract(record.Speciality) + "\n";

count++;

}

}

if (count > 0)

{

return timeRecords;

}

else

{

return "На данный день нет записей";

}

}

/// <summary>

/// Метод вывода количества пациентов у докторов в определённый день.

/// </summary>

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

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

/// <returns></returns>

public static string DoctorRecords(string time, Records[] records)

{

int patients = 0, doctors = 0, diff = 0;

string numberPatients = "";

string[] names = new string[records.Length];

foreach (var record in records)

{

if (record.Time == time)

{

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

{

if (names[i] == record.Initials)

{

diff++;

}

}

if (diff == 0)

{

names[doctors] = record.Initials;

doctors++;

}

}

}

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

{

patients = 0;

foreach (var record in records)

{

if (record.Time == time && record.Initials == names[i])

{

patients++;

}

}

numberPatients = numberPatients + names[i] + ":\n" + patients + "\n\n";

}

if (!string.IsNullOrEmpty(numberPatients))

{

return numberPatients;

}

else

{

return "На данный день нет записей.";

}

}

/// <summary>

/// Метод подсчёта среднего количества пациентов в день по специальностям.

/// </summary>

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

/// <returns></returns>

public static string AveragePatients(Records[] records)

{

double patients = 0, diff = 0;

int days = 0;

string average = "";

string[] times = new string[records.Length];

for (int i = (int)Speciality.Кардиолог; i <= (int)Speciality.Офтальмолог; i++)

{

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

{

times[j] = "";

}

diff = 0;

days = 0;

patients = 0;

foreach (var record in records)

{

if (record.Speciality == i)

{

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

{

if (times[j] == record.Time)

{

diff++;

}

}

if (diff == 0)

{

times[days] = record.Time;

days++;

}

patients++;

}

}

if (days > 0)

{

average = average + "Специальность " + EnumerationExtract(i) + ":\n" + patients / days + "\n\n";

}

}

if (!string.IsNullOrEmpty(average))

{

return average;

}

else

{

return "Записей нет.";

}

}

}

}

**RecordsException.cs:**

namespace LibraryForRecords

{

/// <summary>

/// Класс исключений.

/// </summary>

public class RecordsException : Exception

{

public RecordsException(string message) : base(message)

{

}

/// <summary>

/// Метод исключения для дней.

/// </summary>

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

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

/// <exception cref="MatrixException"></exception>

public static void ValidateDay(string input, out int result)

{

if (string.IsNullOrWhiteSpace(input))

{

throw new RecordsException("Строка пуста!");

}

if (!Int32.TryParse(input, out result))

{

throw new RecordsException("Строка содержит символ или текст!");

}

if (Convert.ToInt32(input) < 0 || Convert.ToInt32(input) > 31 )

{

throw new RecordsException("Не является днём!");

}

}

/// <summary>

/// Метод исключений для месяцев.

/// </summary>

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

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

/// <exception cref="RecordsException"></exception>

public static void ValidateMonth(string input, out int result)

{

if (string.IsNullOrWhiteSpace(input))

{

throw new RecordsException("Строка пуста!");

}

if (!Int32.TryParse(input, out result))

{

throw new RecordsException("Строка содержит символ или текст!");

}

if (Convert.ToInt32(input) < 0 || Convert.ToInt32(input) > 12)

{

throw new RecordsException("Не является месяцом!");

}

}

/// <summary>

/// Метод исключений для годов.

/// </summary>

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

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

/// <exception cref="RecordsException"></exception>

public static void ValidateYear(string input, out int result)

{

if (string.IsNullOrWhiteSpace(input))

{

throw new RecordsException("Строка пуста!");

}

if (!Int32.TryParse(input, out result))

{

throw new RecordsException("Строка содержит символ или текст!");

}

if (Convert.ToInt32(input) < 0)

{

throw new RecordsException("Не является годом!");

}

}

/// <summary>

/// Метод исключения для переменных тип Int.

/// </summary>

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

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

/// <exception cref="MatrixException"></exception>

public static void ValidateInt(string input, out int result)

{

if (string.IsNullOrWhiteSpace(input))

{

throw new RecordsException("Строка пуста!");

}

if (!Int32.TryParse(input, out result))

{

throw new RecordsException("Строка содержит вещественное число или текст!");

}

}

/// <summary>

/// Метод исключения для выбора специальности.

/// </summary>

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

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

/// <exception cref="MatrixException"></exception>

public static void ValidateSpeciality(string input, out int result)

{

if (string.IsNullOrWhiteSpace(input))

{

throw new RecordsException("Строка пуста!\n");

}

if (!Int32.TryParse(input, out result))

{

throw new RecordsException("Строка содержит вещественное число или текст!\n");

}

if (Convert.ToInt32(input) < 1 || Convert.ToInt32(input) > 5)

{

throw new RecordsException("Такой специальности нет!\n");

}

}

}

}

**ПРИЛОЖЕНИЕ В**

(обязательное)

**Текст тестов**

**Tests.cs:**

using LibraryForRecords;

namespace TestsForRecords

{

[TestClass]

public class TestMatrix

{

/// <summary>

/// Тест, проверяющий вывод одной записи.

/// </summary>

[TestMethod]

public void OutputOfOneRecords()

{

RecordsManage records = new RecordsManage(1);

records.records[0].Initials = "renat";

records.records[0].Time = "01.01.2001";

records.records[0].Speciality = 1;

string time = records.records[0].Time;

string record1 = "renat 01.01.2001 Кардиолог\n";

string result = RecordsManage.TimeRecords(time, records.records);

Assert.AreEqual(result, record1);

}

/// <summary>

/// Тест, проверяющий вывод нескольких записей.

/// </summary>

[TestMethod]

public void OutputOfMoreRecords()

{

RecordsManage records = new RecordsManage(3);

records.records[0].Initials = "renat";

records.records[0].Time = "01.01.2001";

records.records[0].Speciality = 1;

records.records[1].Initials = "renat";

records.records[1].Time = "01.01.2001";

records.records[1].Speciality = 2;

records.records[2].Initials = "renat";

records.records[2].Time = "01.01.2001";

records.records[2].Speciality = 3;

string time = records.records[0].Time;

string record1 = "renat 01.01.2001 Кардиолог\nrenat 01.01.2001 Отоларинголог\nrenat 01.01.2001 Психиатр\n";

string result = RecordsManage.TimeRecords(time, records.records);

Assert.AreEqual(result, record1);

}

/// <summary>

/// Тест, проверяющий вывод записи с отличной датой.

/// </summary>

[TestMethod]

public void OutputOfRecordOfDifferentDays()

{

RecordsManage records = new RecordsManage(3);

records.records[0].Initials = "renat";

records.records[0].Time = "01.01.2001";

records.records[0].Speciality = 1;

records.records[1].Initials = "renat";

records.records[1].Time = "01.01.2002";

records.records[1].Speciality = 2;

records.records[2].Initials = "renat";

records.records[2].Time = "01.01.2003";

records.records[2].Speciality = 3;

string time = records.records[0].Time;

string record1 = "renat 01.01.2001 Кардиолог\n";

string result = RecordsManage.TimeRecords(time, records.records);

Assert.AreEqual(result, record1);

}

/// <summary>

/// Тест, проверяющий вывод записей с различной датой.

/// </summary>

[TestMethod]

public void OutupOfRecordsOfDifferentDays()

{

RecordsManage records = new RecordsManage(3);

records.records[0].Initials = "renat";

records.records[0].Time = "01.01.2001";

records.records[0].Speciality = 1;

records.records[1].Initials = "renat";

records.records[1].Time = "01.01.2002";

records.records[1].Speciality = 2;

records.records[2].Initials = "renat";

records.records[2].Time = "01.01.2001";

records.records[2].Speciality = 3;

string time = records.records[0].Time;

string record1 = "renat 01.01.2001 Кардиолог\nrenat 01.01.2001 Психиатр\n";

string result = RecordsManage.TimeRecords(time, records.records);

Assert.AreEqual(result, record1);

}

/// <summary>

/// Тест, проверяющий вывод при отсуствие записи в этот день.

/// </summary>

[TestMethod]

public void OutputWhenNoRecordsOfDay()

{

RecordsManage records = new RecordsManage(3);

records.records[0].Initials = "renat";

records.records[0].Time = "01.01.2001";

records.records[0].Speciality = 1;

records.records[1].Initials = "renat";

records.records[1].Time = "01.01.2002";

records.records[1].Speciality = 2;

records.records[2].Initials = "renat";

records.records[2].Time = "01.01.2001";

records.records[2].Speciality = 3;

string time = "02.02.2001";

string record1 = "На данный день нет записей";

string result = RecordsManage.TimeRecords(time, records.records);

Assert.AreEqual(result, record1);

}

/// <summary>

/// Тест, проверяющий вывод пациента у одного доктора.

/// </summary>

[TestMethod]

public void InHospitalOneDoctorAndOnePatient()

{

RecordsManage records = new RecordsManage(3);

records.records[0].Initials = "renat";

records.records[0].Time = "01.01.2001";

records.records[0].Speciality = 1;

string time = "01.01.2001";

string record1 = "renat:\n1\n\n";

string result = RecordsManage.DoctorRecords(time, records.records);

Assert.AreEqual(result, record1);

}

/// <summary>

/// Тест, проверяющий вывод пациентов у одного доктора.

/// </summary>

[TestMethod]

public void InHospitalOneDoctor()

{

RecordsManage records = new RecordsManage(3);

records.records[0].Initials = "renat";

records.records[0].Time = "01.01.2001";

records.records[0].Speciality = 1;

records.records[1].Initials = "renat";

records.records[1].Time = "01.01.2001";

records.records[1].Speciality = 2;

records.records[2].Initials = "renat";

records.records[2].Time = "01.01.2001";

records.records[2].Speciality = 3;

string time = "01.01.2001";

string record1 = "renat:\n3\n\n";

string result = RecordsManage.DoctorRecords(time, records.records);

Assert.AreEqual(result, record1);

}

/// <summary>

/// Тест, проверяющий вывод пациентов у одного доктора в различные дни.

/// </summary>

[TestMethod]

public void InHospitalOneDoctorInDifferntDays()

{

RecordsManage records = new RecordsManage(3);

records.records[0].Initials = "renat";

records.records[0].Time = "01.01.2001";

records.records[0].Speciality = 1;

records.records[1].Initials = "renat";

records.records[1].Time = "01.01.2002";

records.records[1].Speciality = 2;

records.records[2].Initials = "renat";

records.records[2].Time = "01.01.2001";

records.records[2].Speciality = 3;

string time = "01.01.2001";

string record1 = "renat:\n2\n\n";

string result = RecordsManage.DoctorRecords(time, records.records);

Assert.AreEqual(result, record1);

}

/// <summary>

/// Тест, проверяющий вывод пациентов у нескольких докторов.

/// </summary>

[TestMethod]

public void InHospitalSomeDoctors()

{

RecordsManage records = new RecordsManage(3);

records.records[0].Initials = "renat";

records.records[0].Time = "01.01.2001";

records.records[0].Speciality = 1;

records.records[1].Initials = "ford";

records.records[1].Time = "01.01.2001";

records.records[1].Speciality = 2;

records.records[2].Initials = "renat";

records.records[2].Time = "01.01.2001";

records.records[2].Speciality = 3;

string time = "01.01.2001";

string record1 = "renat:\n2\n\nford:\n1\n\n";

string result = RecordsManage.DoctorRecords(time, records.records);

Assert.AreEqual(result, record1);

}

/// <summary>

/// Тест, проверяющий вывод, когда в этот день нет записей.

/// </summary>

[TestMethod]

public void InThisDayNotRecords()

{

RecordsManage records = new RecordsManage(3);

records.records[0].Initials = "renat";

records.records[0].Time = "01.01.2001";

records.records[0].Speciality = 1;

records.records[1].Initials = "ford";

records.records[1].Time = "01.01.2001";

records.records[1].Speciality = 2;

records.records[2].Initials = "renat";

records.records[2].Time = "01.01.2001";

records.records[2].Speciality = 3;

string time = "01.01.2002";

string record1 = "На данный день нет записей.";

string result = RecordsManage.DoctorRecords(time, records.records);

Assert.AreEqual(result, record1);

}

/// <summary>

/// Тест, проверяющий среднее количество пациентов, когда один пациент в день.

/// </summary>

[TestMethod]

public void OnePatientOnDay()

{

RecordsManage records = new RecordsManage(3);

records.records[0].Initials = "renat";

records.records[0].Time = "01.01.2001";

records.records[0].Speciality = 1;

string record1 = "Специальность Кардиолог:\n1\n\n";

string result = RecordsManage.AveragePatients(records.records);

Assert.AreEqual(result, record1);

}

/// <summary>

/// Тест, проверяющий среднее количество пациентов, когда несколько пациент в день.

/// </summary>

[TestMethod]

public void SomePatientsOnDay()

{

RecordsManage records = new RecordsManage(3);

records.records[0].Initials = "renat";

records.records[0].Time = "01.01.2001";

records.records[0].Speciality = 1;

records.records[1].Initials = "ford";

records.records[1].Time = "01.01.2001";

records.records[1].Speciality = 1;

records.records[2].Initials = "renat";

records.records[2].Time = "01.01.2001";

records.records[2].Speciality = 1;

string record1 = "Специальность Кардиолог:\n3\n\n";

string result = RecordsManage.AveragePatients(records.records);

Assert.AreEqual(result, record1);

}

/// <summary>

/// Тест, проверяющий среднее количество пациентов, когда несколько пациент в различные дни.

/// </summary>

[TestMethod]

public void SomePatientsOnDifferentDays()

{

RecordsManage records = new RecordsManage(3);

records.records[0].Initials = "renat";

records.records[0].Time = "01.01.2001";

records.records[0].Speciality = 1;

records.records[1].Initials = "ford";

records.records[1].Time = "01.01.2002";

records.records[1].Speciality = 1;

records.records[2].Initials = "renat";

records.records[2].Time = "01.01.2003";

records.records[2].Speciality = 1;

string record1 = "Специальность Кардиолог:\n1\n\n";

string result = RecordsManage.AveragePatients(records.records);

Assert.AreEqual(result, record1);

}

/// <summary>

/// Тест, проверяющий среднее количество пациентов, когда несколько пациент по различным специальностям.

/// </summary>

[TestMethod]

public void SomePatientsOfDifferentSpeciality()

{

RecordsManage records = new RecordsManage(3);

records.records[0].Initials = "renat";

records.records[0].Time = "01.01.2001";

records.records[0].Speciality = 1;

records.records[1].Initials = "ford";

records.records[1].Time = "01.01.2002";

records.records[1].Speciality = 2;

records.records[2].Initials = "renat";

records.records[2].Time = "01.01.2003";

records.records[2].Speciality = 5;

string record1 = "Специальность Кардиолог:\n1\n\nСпециальность Отоларинголог:\n1\n\nСпециальность Офтальмолог:\n1\n\n";

string result = RecordsManage.AveragePatients(records.records);

Assert.AreEqual(result, record1);

}

/// <summary>

/// Тест, проверяющий среднее количество пациентов, когда нет записей.

/// </summary>

[TestMethod]

public void NoRecords()

{

RecordsManage records = new RecordsManage(0);

string record1 = "Записей нет.";

string result = RecordsManage.AveragePatients(records.records);

Assert.AreEqual(result, record1);

}

}

}