Министерство просвещения ПМР

ГОУ СПО «Тираспольский техникум информатики и права»

**КУРСОВАЯ РАБОТА**

по дисциплине: Разработка Програмного Обеспечения

на тему: Информационная система поликлиники

Выполнила обучающаяся

Гнатюк Дарья Эдуардовна

Специальность 2.09.02.05

Прикладная информатика

в экономике

Руководитель

преподаватель информационных

дисициплин

Балашова Юлия Владимировна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись руководителя)

Тирасполь, 2021

* **Введение**

Внедрение медицинских информационных систем (МИС) может значительно улучшить и усовершенствовать рабочие процессы в поликлинике. При использовании медицинской информационной системы на базе поликлиники появляется возможность настроить автоматизированное рабочее место врача, создать электронные карты и истории болезни пациентов, облегчить ведение медицинской статистики, автоматизировать административные, хозяйственные, финансовые процессы поликлиники.

Целью данной курсовой работы является изучение возможностей объектно-ориентированной модели программирования в информационной системы поликлиники.

Для выполнения данной курсовой работы были поставлены следующие задачи:

* исследование предметной области, изучение сведений об объектах;
* проектирование программной реализации;
* реализация программного продукта на основе выбранной модели программирования.

Разработать информационную систему поликлиники. Система должна обеспечивать ведение списка новых пациентов, ведение списка пациентов, автоматизированный поиск болезней у пациентов.

Программа должна давать возможность добавления, изменение и удаление данных. Программная реализация должна содержать: интерфейсный класс, абстрактный класс, реальный класс или набор реальных классов, позволяющих решить поставленные задачи.

* **Постановка задачи**
* **Описание предметной области**

Основная деятельность регистратуры поликлиники - это сбор и хранение информации о врачах и пациентах, обработка информации, выдача справок и больничных листов, выдача расписаний работы врачей.

* Информация, хранимая в информационной системе поликлиники:
* Сведения об участках.
* Информация о врачах.
* Сведения о приемах.
* Информация о пациентах (карточка пациента).
* Расписание работы врачей.

Сведения об участках. Участок - это территория, которая относится к поликлинике. Каждый участок обслуживается одним врачом. Один врач может обслуживать несколько участков.

Информация об участке должна содержать:

* номер участка;
* расписание работы участковых врачей;
* информацию о врачах, которые обслуживают участки.

Информация о враче:

* какой врач: участковый, или профильный;
* фамилия;
* имя;
* отчество;
* категория;
* стаж работы;
* дата рождения;
* его участок, если врач участковый;
* специальность.

Сведения о приемах:

* Врач
* Дата приема
* Пациент
* Жалобы
* Диагноз
* Назначение
* Обследование

Информация о пациентах. Информация о пациенте хранится в его карточке. Карточка имеет номер. В карточке указывается:

* Фамилия;
* Имя;
* Отчество пациента;
* Возраст;
* Пол;
* домашний адрес;
* страховой полис.

В карточку заносится информация о каждом посещении поликлиники пациентом: дата посещения, жалобы, предварительный диагноз, назначения, выписан или нет, больничный лист, имя врача и специальность.

Расписание работы врачей. В расписании работы врачей указывается:

* на каком участке работает врач (если врач участковый);
* дни и часы приема;
* номер кабинета:
* специальность.

Регистрация пациентов производится работниками регистратуры, которые заполняют первую страницу карточки. Информацию о болезнях и посещениях вносят врачи. Врач может обслуживать более одного участка. В случае увольнения врача, участок передается другим врачам. Уволить врача может только заведующий поликлиникой. Он удаляет из базы сведения о враче и передает его больных другому врачу.

* **Описание существующих аналогов**

На сегодняшний день существует множество различных медицинских информационных систем. Рассмотрим подробно два вида аналогов.

1С: Медицина. Поликлиника.

Информационная система 1С: Медицина. Поликлиника предназначена для автоматизации основных процессов медицинских организаций различных организационно-правовых форм, оказывающих медицинскую помощь в амбулаторно-поликлинических условиях.

Процессы медицинской организации охватывают многие структурные подразделения, поэтому важно, чтобы автоматизированная система создавала единое информационное пространство медицинской организации. Информационная система «1С: Медицина. Поликлиника» позволяет создать такое информационное пространство с разделением доступа к данным по ролевому принципу. Назначение пользователям ролей позволяет создавать различные автоматизированные места сотрудников.

«1С: Медицина. Поликлиника» может использоваться для учета медицинских услуг и для оперативного планирования деятельности кабинетов медицинской организации, ведения электронных медицинских карт. Оперативное планирование осуществляется с помощью предварительной записи пациентов на выполнение медицинских услуг в медицинских кабинетах. Предварительную запись пациентов может осуществлять как регистратура (регистрационный отдел), так и врачи при выполнении назначений повторных приемов, консультаций, исследований, манипуляций. Для осуществления оперативного планирования врачебному медицинскому персоналу и кабинетам задаются графики работы, нормы загрузки, перечень выполняемых услуг. Услугам, требующим предварительной записи, создаются спецификации (технологические карты), в которых задаются параметры их выполнения.

Информационная система позволяет вести несколько медицинских карт для одного пациента - амбулаторную карту, стоматологическую карту, медицинскую карту больного венерическими заболеваниями и т.д. Количество медицинских карт, а также их типов, не ограничено. Доступ к медицинским картам осуществляется в соответствии с правами доступа. Это значит, что для каждого медицинского работника указывается, к какому типу карт он имеет доступ.

В программе «1С: Медицина. Поликлиника» есть возможность вести учет по нескольким медицинским организациям в одной информационной базе.

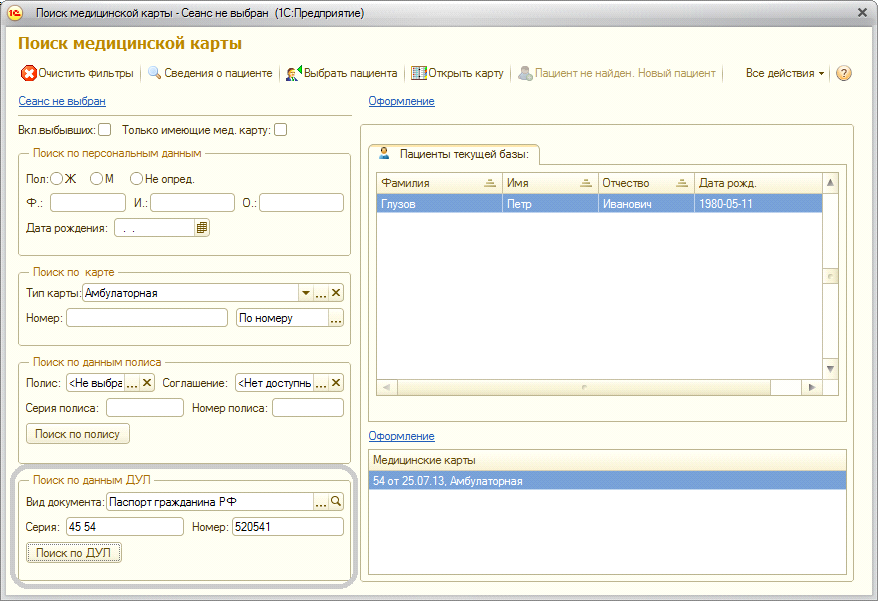


Рисунок 1.1 – 1С: Медицина. Поликлиника

2. Подсистема «Пациент» предназначена для электронной записи на прием к врачу в городе Ухта. Данная подсистема обеспечивает печать взятых номерков. В подсистеме доступен просмотр информации об участковых врачах. Кроме того с помощью данной подсистемы можно узнать график работы врачей, новости поликлиники, в режиме онлайн получить информацию о свободных номерках. В целом подсистема автоматизирует частичную работу регистратуры поликлиники.

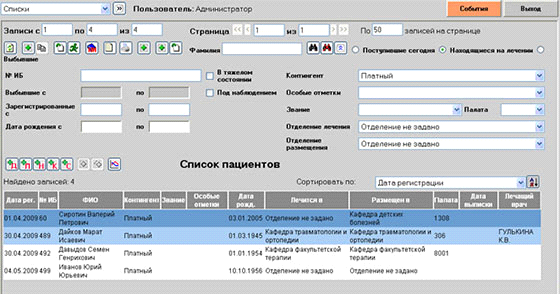


Рисунок 1.2 – Подсистема «Пациент»

Плюсы подсистемы:

* Запись на прием к врачу в режиме онлайн («не выходя из дома»)
* Просмотр информации об участковых врачах
* Просмотр отчета об свободных номерках

Минусы:

* Внедрение в сайт
* Создана только для поликлиники города Ухта
* Основная функция-запись на прием

После изучения существующих аналогов, а также выявленных достоинств и недостатков принято решение о разработке информационной системы поликлиники.

* **Выбор модели реализации программного продукта**

При анализе предметной области и реализации поставленной задачи необходимо сделать выбор модели реализации программного продукта.

Объектно-ориентированное программирование (ООП) – методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

При анализе поставленных задач можно сделать вывод, что для реализации данного программного продукта наилучшим моделью программирования является ООП на языке *С*#.

* **Описание выявленных сущностей**

При анализе предметной области были выявлены следующие сущности:

1. Врач.

2. Пациент.

3. Болезни.

Сущность «Врач» отвечает за информацию о врачах поликлиники, а именно: фамилия, имя, отчество, дата рождения, стаж, специальность. Необходимо реализовать работу с данной информацией, а именно отображение информации, добавление информации, и удаление.

Сущность «Болезни» отвечает за информацию о каждой болезни пациента: название болезни, дата начала болезни, дата окончания болезни. Предусмотреть добавление новой информации, корректировка и удаление. При добавлении информации о контакте и предпочитаемой марки машины предусмотреть добавления нескольких записей.

* **План работы**

Разработку программного продукта было решено разделить на следующие этапы:

* анализ выдвинутых заказчиком требований;
* моделирование теоретической основы проекта;
* работа с кодом: описание структуры классов, поля, методы, свойства, конструкторы, статические поля и методы;
* разработка графического интерфейса программы: формы и элементы управления на них;
* тестирование программы на наличие ошибок.
* исправление найденных ошибок, при тестировании программы.

* **Проектирование и реализация программного   
  продукта**
* **Диаграмма классов**

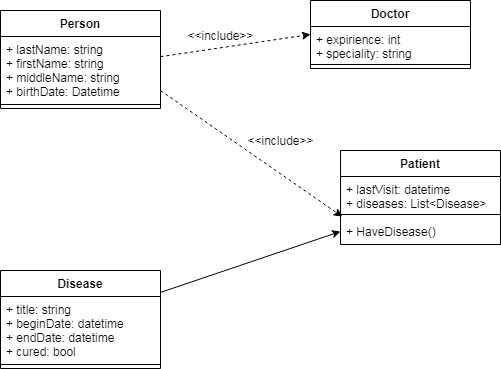


Рисунок 2.1 – Диаграмма классов

Сущность «Пациент» соответствует классу *Person*. В классе реализованы поля и методы, которые были выявлены.

Сущность «Врач» соответствует классу *Doctor* и сущность «Болезнь» соответствует классу *Disease*.

* **Абстрактные классы**

Абстрактный класс – базовый класс, который не предполагает создания экземпляров. Абстрактный класс может содержать (и не содержать) абстрактные методы и свойства. Абстрактный метод не реализуется для класса, в котором описан, однако должен быть реализован для его неабстрактных потомков. В программе реализован абстрактный класс *Person*.

* **Реальные классы**

Класс – некая сущность, которая задает некоторое общее поведение для объектов. Таким образом, любой объект может принадлежать или не принадлежать определенному классу, то есть обладать или не обладать поведением, которое данный класс подразумевает. В программной реализации курсовой работы используются такие классы, как: *Doctor, Patient, Disease.*

Класс *Doctor* используется для описания информации о враче. В классе используются поля: стаж *experience* типа *int*, специальность *specialty* типа *string.* Данный класс содержит метод *ToString().*Класс *Patient* является наследником класса *Person.*

Класс *Patient* используется для описания информации о пациенте. В классе используются поле: последний визит *lastVisit* типа *datetime,* а также предусмотрен список для хранения болезней пациентов*.* Класс *Patient* является наследником класса *Person*, в классе были реализованы следующие методы: для присваивания болезни конкретному пациенту *HaveDisease()*, а также метод *ToString().*

Класс *Disease* описывает информацию о болезни пациента. В классе используются поля: название болезни *title*  типа *string*, дата начала болезни *beginDate* и дата окончания болезни *endDate* типа *datetime*, а также логическая переменная *cured* типа *bool*. В классе был реализован метод *ToString().*

* **Описание программной реализации**

Программный продукт был реализован с использованием объектно-ориентированного программирования, то есть в ней используются классы, каждый из которых описывает некоторую сущность предметной области.

Программная реализация курсовой работы была выполнена в интегрированной среде разработки *Microsoft* *Visual Studio* 2019 на языке программирования высокого уровня *C#*.

* **Руководства пользователя**

Для удобства использования программы пользователю было разработано консольное приложение. После запуска программы пользователю представляется главное окно программы, как показано на рисунке 2.2.

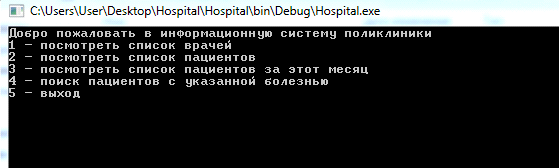


Рисунок 2.2 – Диаграмма классов

Чтобы просмотреть список врачей необходимо ввести с клавиатуры значение – «1», чтобы просмотреть список пациентов необходимо ввести с клавиатуры значение – «2», для просмотра пациентов за текущий месяц необходимо ввести «3», для поиска пациентов с указанной болезнью необходимо ввести значение с клавиатуры число «4», для выхода из приложения необходимо ввести число «5».

При просмотре списка врачей пользователю предлагаются следующие функции (как показано на рисунке 2.3):

* Добавление врача;
* Редактирование информации о враче;
* Удаление информации о враче;
* Возвращение в главное меню.

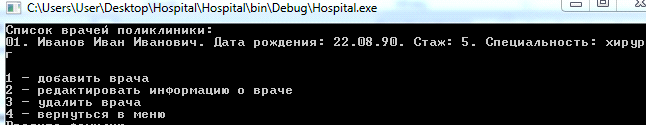


Рисунок 2.3 – Список врачей поликлиники

Для добавления информации о враче необходимо ввести цифру один, а после ввести всю необходимую информацию (рисунок 2.4)

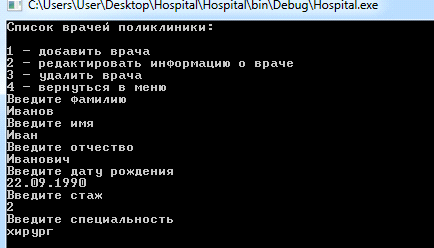
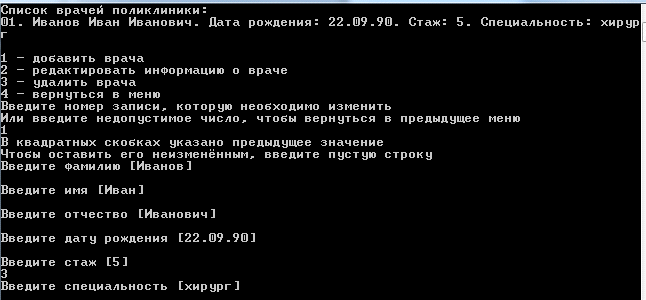


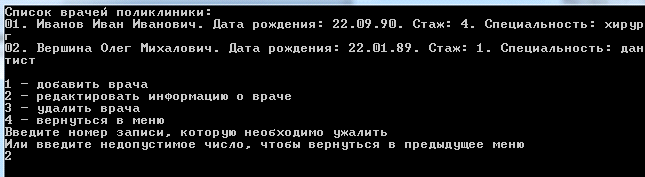
Рисунок 2.4 – Добавление информации и враче

Для редактирование информации о враче необходимо ввести число два, затем ввести номер записи, которую необходимо изменить или ввести недопустимое число чтобы вернуться в предыдущее меню. Затем пользователю необходимо изменить значение в нужном поле и значение будет изменено. Процесс редактирование информации о враче представлена на рисунке 2.5.



* Рисунок 2.5 – Окно редактирование информации о враче

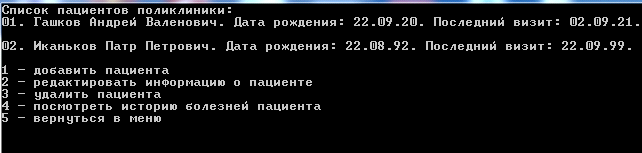
Для удаления информации о враче необходимо ввести цифру три. После пользователю необходимо ввести номер записи. И запись успешно будет удалена, как показано на рисунке 2.6.



* Рисунок 2.6 – Окно удаления информации о враче

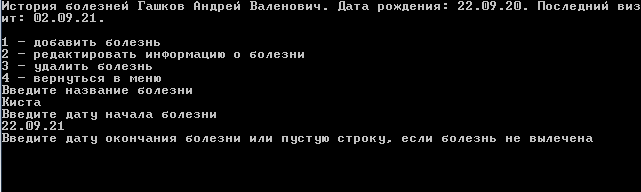
При просмотре списка пациентов поликлиники пользователю предлагаются следующие функции (как показано на рисунке 2.7):

* Добавление пациента;
* Редактирование информации о пациенте;
* Удаление информации о пациенте;
* Просмотр истории болезней пациента;
* Возвращение в главное меню.



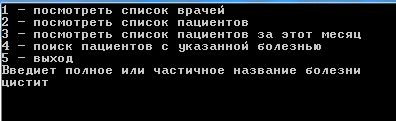
* Рисунок 2.7 – Список пациентов поликлиники

Принцип добавление, редактирования и удаления информации списка пациентов поликлинике аналогичен принципам добавления, редактирования и удаления информации из списка врачей. Пользователю предоставляются вся необходимая информация о пациенте, а также информация о его болезнях. Для добавления информации о болезни человека необходимо ввести цифру один. Затем необходимо ввести название болезни, дату начала болезни и окончания болезни (если таковая имеется, как показана на рисунке 2.8).



* Рисунок 2.8 – Добавления информации о болезни

Чтобы просмотреть список пациентов за этот месяц необходимо выйти в главное меню программы и ввести цифру три. Для поиска пациентов с указанной болезнью необходимо в главном меню программы ввести цифру четыре, затем ввести полное или частичное название болезни, как на рисунке 2.9.



* Рисунок 2.9 – Поиск пациентов по указанной болезни
* **Руководство программиста**

Во время проектирования программного продукта, был создан абстрактный класс *Person*, содержащий открытые поля, конструктор класса и метод *ToString()*. В листинге 3.1 приводится класс *Person*.

|  |
| --- |
| abstract class Person{  public string lastName;  public string firstName;  public string middleName;  public DateTime birthDate;  protected Person() { }  protected Person(string lastName, string firstName, string middleName, DateTime birthDate){  this.lastName = lastName;  this.firstName = firstName;  this.middleName = middleName;  this.birthDate = birthDate;}  public override string ToString(){  return String.Format("{0} {1} {2}. Дата рождения: {3}",  lastName, firstName, middleName, birthDate.ToString("dd.MM.yy"));}} |

1. – Класс *Person*

Класс *Doctor* является наследником абстрактного класса *Person*. В классе реализован метод *ToString()*. В листинге 3.2 приводится код класса *Doctor*.

|  |
| --- |
| class Doctor : Person  {  public int expirience;  public string specialty;  public Doctor() : base() { }  public Doctor(string lastName, string firstName,  string middleName, DateTime birthDate,  int expirience, string specialty) :  base(lastName, firstName, middleName, birthDate)  {  this.expirience = expirience;  this.specialty = specialty;  }  public override string ToString()  {  return String.Format("{0}. Стаж: {1}. Специальность: {2}",  base.ToString(), expirience, specialty);  }  } |

1. – Класс *Doctor*

Класс *Patient* является наследником абстрактного класса *Person*. В классе реализованы методы: *ToString()* – метод формирует строку вывода информации о последнем визите к врачу*,* и метод *HaveDisease()* – осуществляет проверку на наличие болезни для конкретного пациента. В листинге 3.3 приводится реализация кода класса *Patient*.

class Patient : Person

{

public DateTime lastVisit;

public List<Disease> diseases = new List<Disease>();

public Patient() : base() { }

public Patient(string lastName, string firstName,

string middleName, DateTime birthDate,

DateTime lastVisit, params Disease[] diseases) :

base(lastName, firstName, middleName, birthDate)

{

this.lastVisit = lastVisit;

this.diseases.AddRange(diseases);

}

public bool HaveDisease(string disease)

{

return diseases.Any(d => d.title.ToLower().Contains(disease.Trim().ToLower()));

}

public override string ToString()

{

return String.Format("{0}. Последний визит: {1}.",

base.ToString(), lastVisit.ToString("dd.MM.yy"));

}

}

1. – Класс *Patient*

Класс *Disease* является описанием болезней пациентов. В классе реализован метод *ToString()*, который форматирует информацию о болезнях пациента. В листинге 3.4 приводится реализация кода класса *Disease* .

class Disease

{

public string title;

public DateTime beginDate;

public DateTime endDate;

public bool cured;

public Disease() { }

public Disease(string title, DateTime beginDate, DateTime endDate, bool cured = false){

this.title = title;

this.beginDate = beginDate;

this.endDate = endDate;

this.cured = cured;}

public override string ToString()

{

return String.Format("{0}. Дата начала: {1}. {2}",

title, beginDate.ToString("dd.MM.yy"), cured ? "Вылечено: " + endDate.ToString("dd.MM.yy") : "");

}

}

1. – Класс *Disease*

Для работы со списком врачей поликлиники был реализован статический метод *Doctors()*. В методе реализованы вызовы методов для добавление, изменение и удаление информации о врачах в зависимости от введенной цифры пользователем. В листинге 3.5 приводится реализация кода метода *Doctor()*.

static void Doctors(){

bool retry;

do{

Console.Clear();

Console.WriteLine("Список врачей поликлиники: ");

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

Console.WriteLine("{0:D2}. {1}", i + 1, doctors[i].ToString());

Console.WriteLine();

Console.WriteLine("1 - добавить врача");

Console.WriteLine("2 - редактировать информацию о враче");

Console.WriteLine("3 - удалить врача");

Console.WriteLine("4 - вернуться в меню");

do{

retry = false;

switch (Console.ReadKey(true).KeyChar)

{

case '1':

AddDoctor();

break;

case '2':

EditDoctor();

break;

case '3':

DeleteDoctor();

break;

case '4':

return;

default:

retry = true;

break;

}

} while (retry);

} while (true);

}

1. – Метод *Doctor()*

Для добавления информации о врачах был реализован метод *AddDoctor()*. Осуществлен ввод информации о пациентах, а также проверка на ввод записей. В листинге 3.6 приводится реализация кода метода *AddDoctor()*.

static void AddDoctor()

{

Doctor doctor = new Doctor();

AddPerson(doctor);

Console.WriteLine("Введите стаж");

string expirience = Console.ReadLine();

while (!int.TryParse(expirience, out doctor.expirience))

{

Console.WriteLine("Ошибка ввода. Повторите");

expirience = Console.ReadLine();

}

Console.WriteLine("Введите специальность");

doctor.specialty = Console.ReadLine();

while (string.IsNullOrWhiteSpace(doctor.specialty))

{

Console.WriteLine("Ошибка ввода. Повторите");

doctor.specialty = Console.ReadLine();

}

doctors.Add(doctor);

}

}

1. – Метод *AddDoctor()*

Для редактирования данных о врачах был реализован статический метод *EditDoctor().* По введенному номеру записи для изменения происходит поиск, и если запись существует происходит пошаговое изменение. Если записи не существует программа выходит из метода. Пример кода показан на листинге 3.7.

static void EditDoctor(){

Console.WriteLine("Введите номер записи, которую необходимо изменить");

Console.WriteLine("Или введите недопустимое число, чтобы вернуться в предыдущее меню");

int i;

if (!int.TryParse(Console.ReadLine(), out i) || i <= 0 || i > doctors.Count)return;i--;

EditPerson(doctors[i]);

Console.WriteLine("Введите стаж [{0}]", doctors[i].expirience);

string expirience = Console.ReadLine();

int exp;

while (!int.TryParse(expirience, out exp) && !string.IsNullOrWhiteSpace(expirience)){

Console.WriteLine("Ошибка ввода. Повторите");

expirience = Console.ReadLine();}

if (!string.IsNullOrWhiteSpace(expirience))

doctors[i].expirience = exp;

Console.WriteLine("Введите специальность [{0}]", doctors[i].specialty);

string specialty = Console.ReadLine();

if (!string.IsNullOrWhiteSpace(specialty))

doctors[i].specialty = specialty;}}}

Листинг 3.7 – Метод *EditDoctor()*

Изменение информации о болезнях и пациентах реализованы аналогичным способам, как рассмотрено выше на примерах.

* **Заключение**

В данной курсовой работе была изучена предметная область информационной системы поликлинике. При анализе предметной области и реализации поставленной задачи была выбрана модель реализации программного продукта, а именно объектно-ориентированной программирование. Выявлены сущности и взаимодействия между ними.

В ходе выполнения курсовой работы была исследована предметная область курсовой работы, продумана постановка задач, построена диаграмма классов, а также спроектировано и реализовано требуемое программное обеспечение. Для программной реализации были выполнены поставленные задачи: разработана информационная система поликлиники. Система обеспечивает ведение списка пациентов (фамилия, имя, отчество, дата рождения, последнее обращение в поликлинику), списка врачей (фамилия, имя, отчество, дата рождения, должность, стаж, образование), ведение списка болезней пациентов (название болезни, дата начала болезни, дата окончания болезни, если пациент завершил лечение).

Программа дает возможность пользователю добавление, изменение и удаление данных.

* **Список использованной литературы**

1. Шилдт Г. Полный справочник по *С#*. : Пер. с англ. – М. : Издательский дом «Вильямс», 2004.

2. Рихтер Джефри. Программирование на платформе *Microsoft .NET Framework 2.0* на языке *C#*. Мастер класс./Пер. с англ. – 2-е изд., испр. – М.: Издательско-торговый дом «Русская Редакция», 2007.

3. Павловская Т.А*. С#.* Программирование на языке высокого уровня. Учебник для вузов. – СПб.: Питер, 2007.

4. Либерти Джесс. Программирование на *С#*.: Пер. с англ. – 2-е изд., С.Пб.: Издательство «Символ-плюс», 2002.

5.Уроки по *С#* [Электронный ресурс]. – URL: [http://www.programbeginner.ru](http://www.programbeginner.ru/).

6. *Visual C#* для чайников [Электронный ресурс ]. – URL: http://easyprog.ru (дата обращение 24.05.19).

7. Сайт о программировании [Электронный ресурс ]. – URL: https://metanit.com

* **Приложение А**

|  |
| --- |
| abstract class EnumString  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Hospital  {  class Program  {  static List<Doctor> doctors = new List<Doctor>();  static List<Patient> patients = new List<Patient>();  static void Main(string[] args)  {  bool retry;  Console.WriteLine("Добро пожаловать в информационную систему поликлиники");  do  {  Console.WriteLine("1 - посмотреть список врачей");  Console.WriteLine("2 - посмотреть список пациентов");  Console.WriteLine("3 - посмотреть список пациентов за этот месяц");  Console.WriteLine("4 - поиск пациентов с указанной болезнью");  Console.WriteLine("5 - выход");  do  {  retry = false;  switch (Console.ReadKey(true).KeyChar)  {  case '1':  Doctors();  break;  case '2':  Patients();  break;  case '3':  foreach(var p in patients.Where(i => i.lastVisit.Month == DateTime.Now.Month))  Console.WriteLine(p.ToString());  Console.ReadKey(true);  break;  case '4':  Console.WriteLine("Введиет полное или частичное название болезни"); ;  string d = Console.ReadLine();  foreach (var p in patients.Where(i => i.HaveDisease(d)))  Console.WriteLine(p.ToString());  Console.ReadKey(true);  break;  case '5':  return;  default:  retry = true;  break;  }  } while (retry);  Console.Clear();  } while (true);  }  static void Doctors()  {  bool retry;  do  {  Console.Clear();  Console.WriteLine("Список врачей поликлиники: ");  for (int i = 0; i < doctors.Count; i++)  Console.WriteLine("{0:D2}. {1}", i + 1, doctors[i].ToString());  Console.WriteLine();  Console.WriteLine("1 - добавить врача");  Console.WriteLine("2 - редактировать информацию о враче");  Console.WriteLine("3 - удалить врача");  Console.WriteLine("4 - вернуться в меню");  do  {  retry = false;  switch (Console.ReadKey(true).KeyChar)  {  case '1':  AddDoctor();  break;  case '2':  EditDoctor();  break;  case '3':  DeleteDoctor();  break;  case '4':  return;  default:  retry = true;  break;  }  } while (retry);  } while (true);  }  static void Patients()  {  bool retry;  do  {  Console.Clear();  Console.WriteLine("Список пациентов поликлиники: ");  for (int i = 0; i < patients.Count; i++)  Console.WriteLine("{0:D2}. {1}", i + 1, patients[i].ToString());  Console.WriteLine();  Console.WriteLine("1 - добавить пациента");  Console.WriteLine("2 - редактировать информацию о пациенте");  Console.WriteLine("3 - удалить пациента");  Console.WriteLine("4 - посмотреть историю болезней пациента");  Console.WriteLine("5 - вернуться в меню");  do  {  retry = false;  switch (Console.ReadKey(true).KeyChar)  {  case '1':  AddPatient();  break;  case '2':  EditPatient();  break;  case '3':  DeletePatient();  break;  case '4':  Diseases();  break;  case '5':  return;  default:  retry = true;  break;  }  } while (retry);  } while (true);  }  static void AddPerson(Person p)  {  Console.WriteLine("Введите фамилию");  p.lastName = Console.ReadLine();  while(string.IsNullOrWhiteSpace(p.lastName))  {  Console.WriteLine("Ошибка ввода. Повторите");  p.lastName = Console.ReadLine();  }  Console.WriteLine("Введите имя");  p.firstName = Console.ReadLine();  while (string.IsNullOrWhiteSpace(p.firstName))  {  Console.WriteLine("Ошибка ввода. Повторите");  p.firstName = Console.ReadLine();  }  Console.WriteLine("Введите отчество");  p.middleName = Console.ReadLine();  while (string.IsNullOrWhiteSpace(p.middleName))  {  Console.WriteLine("Ошибка ввода. Повторите");  p.middleName = Console.ReadLine();  }  Console.WriteLine("Введите дату рождения");  string birth = Console.ReadLine();  while (!DateTime.TryParse(birth, out p.birthDate))  {  Console.WriteLine("Ошибка ввода. Повторите");  birth = Console.ReadLine();  }  }  static void EditPerson(Person p)  {  Console.WriteLine("В квадратных скобках указано предыдущее значение");  Console.WriteLine("Чтобы оставить его неизменённым, введите пустую строку");  Console.WriteLine("Введите фамилию [{0}]", p.lastName);  string lastName = Console.ReadLine();  if (!string.IsNullOrWhiteSpace(lastName))  p.lastName = lastName;  Console.WriteLine("Введите имя [{0}]", p.firstName);  string firstName = Console.ReadLine();  if (!string.IsNullOrWhiteSpace(firstName))  p.firstName = firstName;  Console.WriteLine("Введите отчество [{0}]", p.middleName);  string middleName = Console.ReadLine();  if (!string.IsNullOrWhiteSpace(middleName))  p.middleName = middleName;  Console.WriteLine("Введите дату рождения [{0}]", p.birthDate.ToString("dd.MM.yy"));  string birth = Console.ReadLine();  DateTime birthDate;  while (!DateTime.TryParse(birth, out birthDate) && !string.IsNullOrWhiteSpace(birth))  {  Console.WriteLine("Ошибка ввода. Повторите");  birth = Console.ReadLine();  }  if (!string.IsNullOrWhiteSpace(birth))  p.birthDate = birthDate;  }  static void AddDoctor()  {  Doctor doctor = new Doctor();  AddPerson(doctor);  Console.WriteLine("Введите стаж");  string expirience = Console.ReadLine();  while (!int.TryParse(expirience, out doctor.expirience))  {  Console.WriteLine("Ошибка ввода. Повторите");  expirience = Console.ReadLine();  }  Console.WriteLine("Введите специальность");  doctor.specialty = Console.ReadLine();  while (string.IsNullOrWhiteSpace(doctor.specialty))  {  Console.WriteLine("Ошибка ввода. Повторите");  doctor.specialty = Console.ReadLine();  }  doctors.Add(doctor);  }  static void EditDoctor()  {  Console.WriteLine("Введите номер записи, которую необходимо изменить");  Console.WriteLine("Или введите недопустимое число, чтобы вернуться в предыдущее меню");  int i;  if (!int.TryParse(Console.ReadLine(), out i) || i <= 0 || i > doctors.Count)  return;  i--;  EditPerson(doctors[i]);  Console.WriteLine("Введите стаж [{0}]", doctors[i].expirience);  string expirience = Console.ReadLine();  int exp;  while (!int.TryParse(expirience, out exp) && !string.IsNullOrWhiteSpace(expirience))  {  Console.WriteLine("Ошибка ввода. Повторите");  expirience = Console.ReadLine();  }  if (!string.IsNullOrWhiteSpace(expirience))  doctors[i].expirience = exp;  Console.WriteLine("Введите специальность [{0}]", doctors[i].specialty);  string specialty = Console.ReadLine();  if (!string.IsNullOrWhiteSpace(specialty))  doctors[i].specialty = specialty;  }  static void DeleteDoctor()  {  Console.WriteLine("Введите номер записи, которую необходимо ужалить");  Console.WriteLine("Или введите недопустимое число, чтобы вернуться в предыдущее меню");  int i;  if (!int.TryParse(Console.ReadLine(), out i) || i <= 0 || i > doctors.Count)  return;  doctors.RemoveAt(i - 1);  }  static void AddPatient()  {  Patient patient = new Patient();  AddPerson(patient);  Console.WriteLine("Введите дату последнего визита");  string visit = Console.ReadLine();  while (!DateTime.TryParse(visit, out patient.lastVisit))  {  Console.WriteLine("Ошибка ввода. Повторите");  visit = Console.ReadLine();  }  patients.Add(patient);  }  static void EditPatient()  {  Console.WriteLine("Введите номер записи, которую необходимо изменить");  Console.WriteLine("Или введите недопустимое число, чтобы вернуться в предыдущее меню");  int i;  if (!int.TryParse(Console.ReadLine(), out i) || i <= 0 || i > patients.Count)  return;  i--;  EditPerson(patients[i]);  Console.WriteLine("Введите дату последнего визита [{0}]", patients[i].lastVisit.ToString("dd.MM.yy"));  string visit = Console.ReadLine();  DateTime lastVisit;  while (!DateTime.TryParse(visit, out lastVisit) && !string.IsNullOrWhiteSpace(visit))  {  Console.WriteLine("Ошибка ввода. Повторите");  visit = Console.ReadLine();  }  if (!string.IsNullOrWhiteSpace(visit))  patients[i].lastVisit = lastVisit;  }  static void DeletePatient()  {  Console.WriteLine("Введите номер записи, которую необходимо ужалить");  Console.WriteLine("Или введите недопустимое число, чтобы вернуться в предыдущее меню");  int i;  if (!int.TryParse(Console.ReadLine(), out i) || i <= 0 || i > patients.Count)  return;  patients.RemoveAt(i - 1);  }  static void Diseases()  {  Console.WriteLine("Введите номер пациента, историю болезней которого необходимо показать");  Console.WriteLine("Или введите недопустимое число, чтобы вернуться в предыдущее меню");  int i;  if (!int.TryParse(Console.ReadLine(), out i) || i <= 0 || i > patients.Count)  return;  i--;  bool retry;  do  {  Console.Clear();  Console.WriteLine("История болезней {0}", patients[i].ToString());  for (int j = 0; j < patients[i].diseases.Count; j++)  Console.WriteLine("{0:D2}. {1}", j + 1, patients[i].diseases[j].ToString());  Console.WriteLine();  Console.WriteLine("1 - добавить болезнь");  Console.WriteLine("2 - редактировать информацию о болезни");  Console.WriteLine("3 - удалить болезнь");  Console.WriteLine("4 - вернуться в меню");  do  {  retry = false;  switch (Console.ReadKey(true).KeyChar)  {  case '1':  AddDisease(patients[i]);  break;  case '2':  EditDisease(patients[i]);  break;  case '3':  DeleteDisease(patients[i]);  break;  case '4':  return;  default:  retry = true;  break;  }  } while (retry);  } while (true);  }  static void AddDisease(Patient p)  {  Disease disease = new Disease();  Console.WriteLine("Введите название болезни");  disease.title = Console.ReadLine();  while (string.IsNullOrWhiteSpace(disease.title))  {  Console.WriteLine("Ошибка ввода. Повторите");  disease.title = Console.ReadLine();  }  Console.WriteLine("Введите дату начала болезни");  string begin = Console.ReadLine();  while (!DateTime.TryParse(begin, out disease.beginDate))  {  Console.WriteLine("Ошибка ввода. Повторите");  begin = Console.ReadLine();  }  Console.WriteLine("Введите дату окончания болезни или пустую строку, если болезнь не вылечена");  string end = Console.ReadLine();  DateTime endDate;  while (!DateTime.TryParse(end, out endDate) && !string.IsNullOrWhiteSpace(end))  {  Console.WriteLine("Ошибка ввода. Повторите");  end = Console.ReadLine();  }  if (!string.IsNullOrWhiteSpace(end))  {  disease.endDate = endDate;  disease.cured = true;  }  p.diseases.Add(disease);  }  static void EditDisease(Patient p)  {  Console.WriteLine("Введите номер записи, которую необходимо изменить");  Console.WriteLine("Или введите недопустимое число, чтобы вернуться в предыдущее меню");  int i;  if (!int.TryParse(Console.ReadLine(), out i) || i <= 0 || i > patients.Count)  return;  i--;  Console.WriteLine("Введите специальность [{0}]", p.diseases[i].title);  string title = Console.ReadLine();  if (!string.IsNullOrWhiteSpace(title))  p.diseases[i].title = title;  Console.WriteLine("Введите дату начала болезни [{0}]", p.diseases[i].beginDate.ToString("dd.MM.yy"));  string begin = Console.ReadLine();  DateTime beginDate;  while (!DateTime.TryParse(begin, out beginDate) && !string.IsNullOrWhiteSpace(begin))  {  Console.WriteLine("Ошибка ввода. Повторите");  begin = Console.ReadLine();  }  if (!string.IsNullOrWhiteSpace(begin))  p.diseases[i].beginDate = beginDate;  Console.WriteLine("Была ли вылечена болезнь [{0}]", p.diseases[i].cured ? "да" : "нет");  string cured = Console.ReadLine().Trim().ToLower();  while (cured != "да" && cured != "нет" && !string.IsNullOrWhiteSpace(cured))  {  Console.WriteLine("Ошибка ввода. Повторите");  cured = Console.ReadLine().Trim().ToLower();  }  bool old = p.diseases[i].cured;  if (!string.IsNullOrWhiteSpace(cured))  {  if (cured == "да")  p.diseases[i].cured = true;  else  p.diseases[i].cured = false;  }  if (!p.diseases[i].cured)  return;  Console.WriteLine("Введите дату окончания болезни [{0}]", old ? p.diseases[i].endDate.ToString("dd.MM.yy") : "");  string end = Console.ReadLine();  DateTime endDate;  while (!DateTime.TryParse(end, out endDate) && !string.IsNullOrWhiteSpace(end))  {  Console.WriteLine("Ошибка ввода. Повторите");  begin = Console.ReadLine();  }  if (!string.IsNullOrWhiteSpace(end))  p.diseases[i].endDate = endDate;  }  static void DeleteDisease(Patient p)  {  Console.WriteLine("Введите номер записи, которую необходимо ужалить");  Console.WriteLine("Или введите недопустимое число, чтобы вернуться в предыдущее меню");  int i;  if (!int.TryParse(Console.ReadLine(), out i) || i <= 0 || i > p.diseases.Count)  return;  p.diseases.RemoveAt(i - 1);  }  }  }}} |

* **Приложение Б**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Hospital

{

abstract class Person

{

public string lastName;

public string firstName;

public string middleName;

public DateTime birthDate;

protected Person() { }

protected Person(string lastName, string firstName, string middleName, DateTime birthDate)

{

this.lastName = lastName;

this.firstName = firstName;

this.middleName = middleName;

this.birthDate = birthDate;

}

public override string ToString()

{

return String.Format("{0} {1} {2}. Дата рождения: {3}",

lastName, firstName, middleName, birthDate.ToString("dd.MM.yy"));

}

}

class Doctor : Person

{

public int expirience;

public string specialty;

public Doctor() : base() { }

public Doctor(string lastName, string firstName,

string middleName, DateTime birthDate,

int expirience, string specialty) :

base(lastName, firstName, middleName, birthDate)

{

this.expirience = expirience;

this.specialty = specialty;

}

public override string ToString()

{

return String.Format("{0}. Стаж: {1}. Специальность: {2}",

base.ToString(), expirience, specialty);

}

}

class Patient : Person

{

public DateTime lastVisit;

public List<Disease> diseases = new List<Disease>();

public Patient() : base() { }

public Patient(string lastName, string firstName,

string middleName, DateTime birthDate,

DateTime lastVisit, params Disease[] diseases) :

base(lastName, firstName, middleName, birthDate)

{

this.lastVisit = lastVisit;

this.diseases.AddRange(diseases);

}

public bool HaveDisease(string disease)

{

return diseases.Any(d => d.title.ToLower().Contains(disease.Trim().ToLower()));

}

public override string ToString()

{

return String.Format("{0}. Последний визит: {1}.",

base.ToString(), lastVisit.ToString("dd.MM.yy"));

}

}

class Disease

{

public string title;

public DateTime beginDate;

public DateTime endDate;

public bool cured;

public Disease() { }

public Disease(string title, DateTime beginDate, DateTime endDate, bool cured = false)

{

this.title = title;

this.beginDate = beginDate;

this.endDate = endDate;

this.cured = cured;

}

public override string ToString()

{

return String.Format("{0}. Дата начала: {1}. {2}",

title, beginDate.ToString("dd.MM.yy"), cured ? "Вылечено: " + endDate.ToString("dd.MM.yy") : "");

}

}

}}}