



Министерство образования и науки Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение высшего  
профессионального образования  
«Санкт-Петербургский государственный электротехнический университет  
“ЛЭТИ” им.В.И.Ульянова (Ленина)»

Кафедра ВТ

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ  
ПРОЕКТУ ПО ДИСЦИПЛИНЕ «ОБЪЕКТНО-  
ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ»**

**Тема: «Создание программного комплекса средствами объектно-  
ориентированного программирования»**

Студент гр. 3311

Локтионов Т.И.

Преподаватель

Павловский М.Г

Санкт-Петербург

2024

## **Введение:**

Данная курсовая работа посвящена разработке программного комплекса (ПК) для управления деятельностью клиники. Программа предоставляет удобный интерфейс для ведения сведений о пациентах, врачах, расписании приёмов, а также для формирования различных отчётов и статистики. Основной целью является упрощение работы менеджера клиники за счёт автоматизации ввода данных, улучшения процесса хранения информации и обеспечения доступа к аналитическим результатам.

**Обязательными требованиями при разработке кода ПК являются использование следующих конструкций языка Java:**

- закрытые и открытые члены классов;
- наследование;
- конструкторы с параметрами;
- абстрактные базовые классы;
- виртуальные функции;
- обработка исключительных ситуаций;
- динамическое создание объектов.

## **Техническое задание:**

Задание 8. Разработать ПК для администратора регистратуры поликлиники. В ПК должна храниться информация об больных, врачах и их расписании работы. Администратор регистратуры может добавлять, изменять и удалять эту информацию. Ему могут потребоваться следующие сведения:

состав гонщиков команды;

- Список врачей и их специализация
- График работы врачей (реализован как дата записи)
- Справка о болезни некоторого больного
- Вид болезни

### **Технические возможности ПК:**

1. Управление данными о пациентах с возможностью добавления, редактирования и удаления записей.
2. Управление сведениями о врачах, включая указание специализации и назначение пациентов.
3. Формирование расписания приёмов, в том числе даты, времени и статуса (принят, отменён, в ожидании).
4. Отображение списка врачей и привязанных к ним пациентов.
5. Генерация информации о статусах приёмов и итогах лечения для каждого пациента.
6. Создание и редактирование графика приёмов, позволяющее оптимизировать загрузку врачей.
7. Вывод и сортировка данных о приёмах с возможностью фильтрации по дате, врачу или статусу.
8. Экспорт данных в форматы PDF и HTML (с использованием JasperReports).
9. Поддержка работы с XML-файлами для сохранения и загрузки информации о пациентах и приёмах.

### **Описание Программного Комплекса:**

Программный комплекс реализован на языке Java и представляет собой настольное приложение с графическим интерфейсом для управления данными клиники. В нём используются компоненты JTable и DefaultTableModel для хранения и отображения сведений о пациентах, врачах и приёмах. Ключевые возможности включают многопоточную обработку, а также экспорт данных в PDF и HTML. Интерфейс приложения содержит элементы для добавления, редактирования и удаления записей, а также инструменты фильтрации и сортировки, что упрощает и ускоряет работу персонала клиники.

### **Проектирование ПК**

На этапе проектирования системы для управления информацией о пациентах и приёмах необходимо было определить основные требования и функциональные сценарии. Каждый сценарий описывает, какие действия пользователь может выполнять и как приложение реагирует на эти действия. Эти сценарии (прецеденты, use cases) дают чёткое понимание ключевых процессов в клинике и упрощают реализацию системы.

#### **Прецеденты**

Добавить пациента: пользователь (например, регистратор) вносит нового пациента в базу, используя данные, загруженные из XML-файла или введённые вручную.

Удалить пациента: пользователь удаляет данные о конкретном пациенте.

Сохранить данные: пользователь сохраняет текущие записи (пациенты, приёмы и т.д.) в XML-файл.

Загрузить данные: пользователь загружает список пациентов и другую информацию из существующего XML-файла.

Поиск пациента: пользователь ищет пациента по ключевым словам (например, имени, врачу или названию болезни).

Регистрация пользователя: новый пользователь (сотрудник клиники) получает

учётные данные для входа в систему.

Генерация отчётов: пользователь формирует отчёты в PDF или HTML-формате, используя данные о пациентах, датах приёма или статусах.

### **Акторы**

Пользователь: основной актор, который работает с данными (вносит пациентов, ищет, редактирует записи).

Администратор: актор с расширенными правами, который управляет доступом к системе (например, регистрирует новых сотрудников).

### **Связи между актёрами и прецедентами**

На диаграмме прецедентов пользователь инициирует все основные операции с данными (добавление, удаление, поиск и т.д.).

Администратор управляет правами пользователей и процессом регистрации — это может быть отображено отдельной стрелкой к прецеденту «Регистрация пользователя».

### **Связи использования и расширения**

Связь *uses* (использование) отражает ситуацию, когда один прецедент необходим другому. К примеру, «Сохранить данные» может включать повторное использование логики «Загрузить данные» для валидации или проверки целостности.

Связь *extends* (расширение) показывает, что один прецедент дополняет другой дополнительным поведением. Например, «Поиск пациента» может быть расширён («extends») функциональностью более сложной фильтрации или сортировки.

### **Ранжирование прецедентов**

К критически важным функциям относятся «Добавить пациента» и «Удалить пациента», так как без них базовая работа с системой невозможна.

«Сохранить данные» и «Загрузить данные» также входят в число приоритетных, обеспечивая целостность информации.

«Поиск пациента» и «Генерация отчётов» могут идти следом по важности, поскольку они значительно облегчают анализ и доступ к данным.

### **Элементы диаграммы прецедентов**

Акторы: «Пользователь» (персонал клиники) и «Администратор» (управляет доступом и настройками).

Прецеденты: «Добавить пациента», «Удалить пациента», «Сохранить данные», «Загрузить данные», «Поиск пациента», «Регистрация пользователя», «Генерация отчётов».

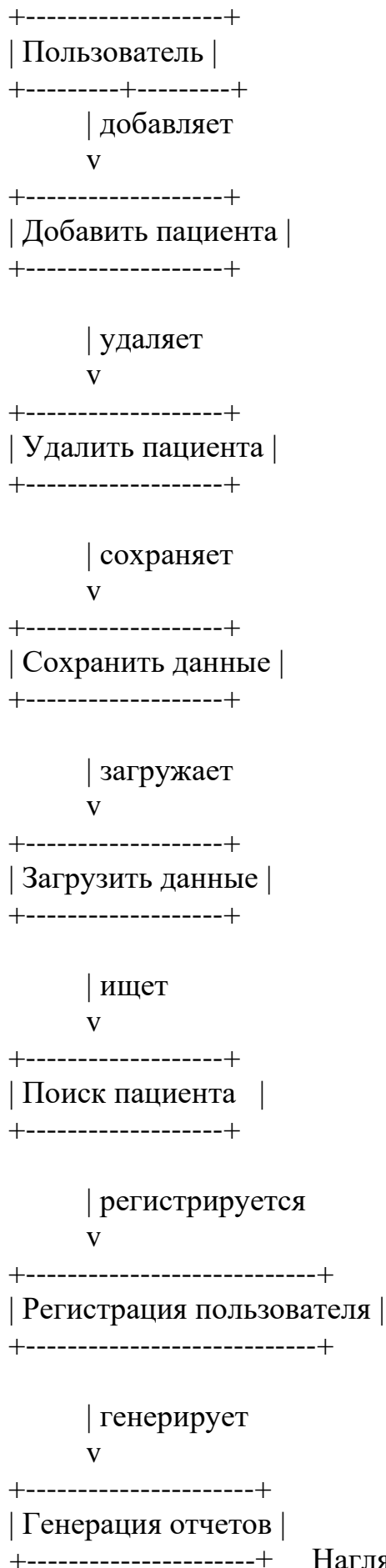
Связи: стрелки между актёрами и прецедентами, а также *uses/extends*, если один прецедент опирается на функциональность другого.

### **Описание диаграммы**

«Пользователь» связан со всеми основными действиями по работе с пациентами (добавление, удаление, поиск), а также с генерацией отчётов.

«Администратор» выполняет регистрацию новых пользователей, контролирует их права.

«Сохранить данные» и «Загрузить данные» логически связаны, так как оба обращаются к работе с XML-файлами.



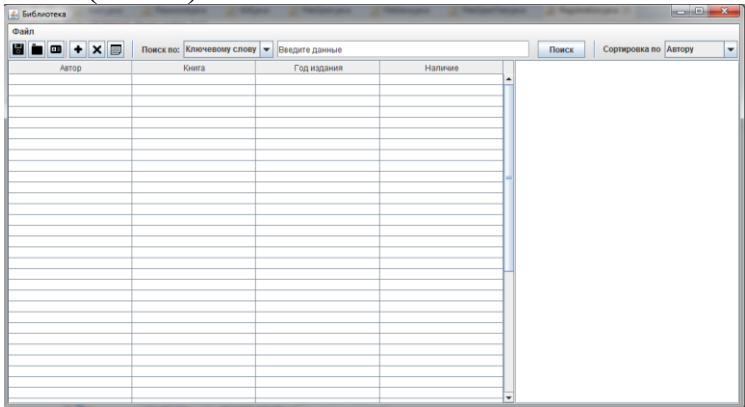
Наглядная диаграмма прецедентов отражает эти взаимосвязи, позволяя в дальнейшем легко проектировать и реализовывать логику приложения.

### Создание прототипа интерфейса пользователя

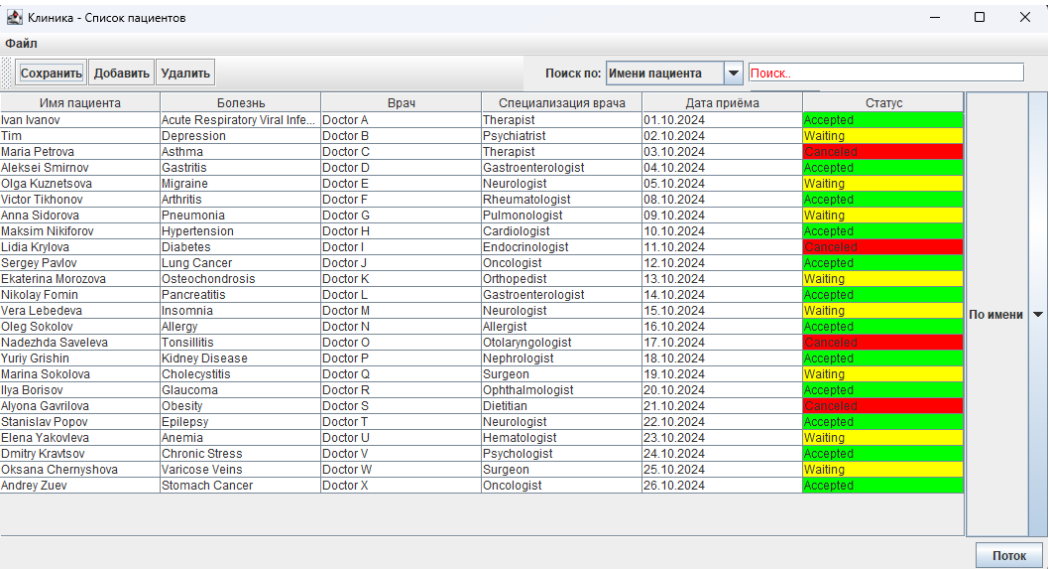
Прецеденты описывают ключевой функционал системы «в общих чертах», не вдаваясь в особенности реализации пользовательского интерфейса. Для разработки же удобного и понятного интерфейса необходимо детально прописать, **какие именно формы** (диалоговые окна, экраны) будут доступны пользователю, **какие поля ввода** и **элементы управления** (кнопки, пункты меню и т.д.) на них расположены, а также **какие действия** требуется совершать (нажатие кнопки, ввод текста, выбор из выпадающего списка и пр.) и **какую реакцию** (вывод сведений, отображение подсказки, перемещение курсора) система при этом даёт.

### Создание прототипа интерфейса пользователя:

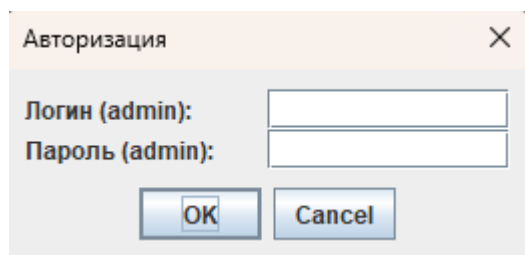
Главное окно (макет):



Готовый результат:



Регистрация пользователя (авторизация):



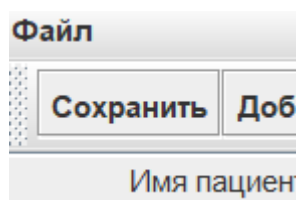
Авторизация

Логин (admin):

Пароль (admin):

OK Cancel

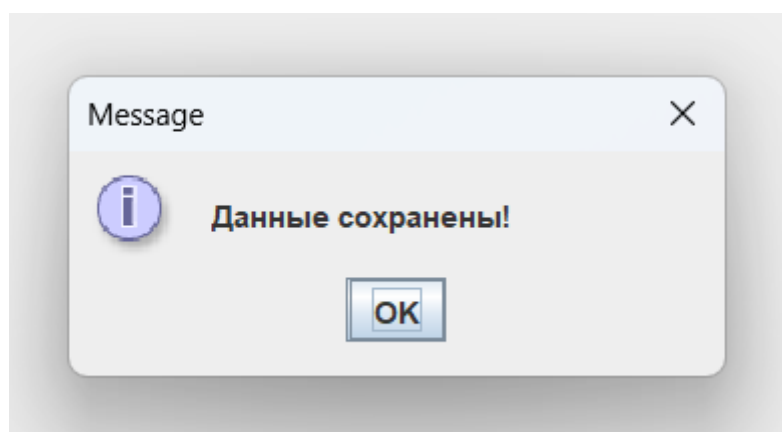
Сохранение данных:



Файл

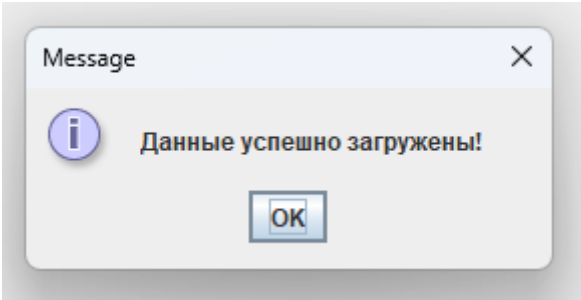
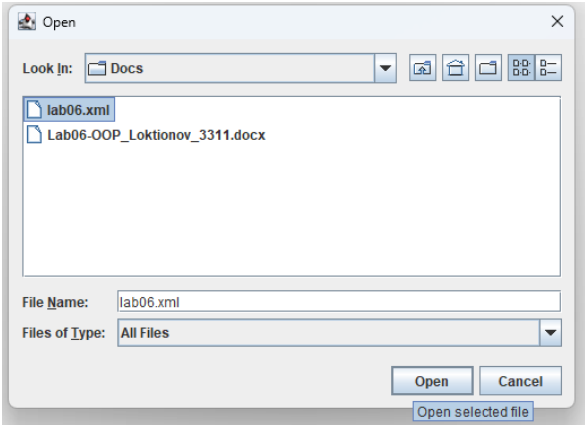
Сохранить Доб

Имя пациен



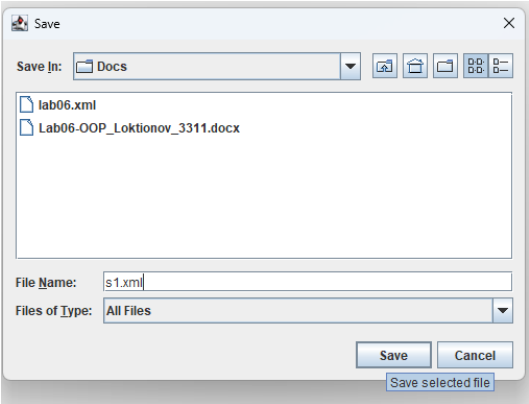
Работоспособность загрузки/выгрузки данных из XML файла:

```
<patients>
  <patient>
    <name>Иван Иванов</name>
    <disease>ОРВИ</disease>
    <doctor>Доктор А</doctor>
    <specialization>Терапевт</specialization>
    <date>01.10.2024</date>
    <status>Принят</status>
  </patient>
  <patient>
    <name>ティム</name>
    <disease>幸せになりたい</disease>
    <doctor>Доктор Б</doctor>
    <specialization>Психиатр</specialization>
    <date>02.10.2024</date>
    <status>Ожидание</status>
  </patient>
  <patient>
    <name>Мария Петрова</name>
    <disease>アスピラマ</disease>
    <doctor>Доктор В</doctor>
    <specialization>Терапевт</specialization>
    <date>03.10.2024</date>
    <status>Отменен</status>
  </patient>
  <patient>
    <name>Алексей Смирнов</name>
    <disease>Гастрит</disease>
    <doctor>ドクター G</doctor>
    <specialization>消化器内科医</specialization>
    <date>04.10.2024</date>
    <status>Принят</status>
  </patient>
  <patient>
    <name>Ольга Кузнецова</name>
    <disease>Мигрень</disease>
    <doctor>Доктор Д</doctor>
    <specialization>Невролог</specialization>
    <date>05.10.2024</date>
    <status>Ожидание</status>
  </patient>
</patients>
```



Имя пациента	Болезнь	Врач	
Иван Иванов	ОРВИ	Доктор А	Тер
ティム	幸せになりたい	Доктор Б	Пси
Мария Петрова	アスピラマ	Доктор В	Тер
Алексей Смирнов	Гастрит	ドクター G	消化
Ольга Кузнецова	Мигрень	Доктор Д	Ней
グイクトルティホノフ	アトリット	Доктор З	Реч

Имя пациента	Болезнь	Врач
Иван Иванов	ОРВИ	Доктор А



```
<patients>
  <patient>
    <name>Иван Иванов</name>
    <disease>ОРВИ</disease>
    <doctor>Доктор А</doctor>
    <specialization>Терапевт</specialization>
    <date>01.10.2024</date>
    <status>Принят</status>
  </patient>
</patients>
```



Сортировка:

Имя пациента ▾	
Yuriy Grishin	Kidney
Victor Tikhonov	Arthritis
Vera Lebedeva	Insomr
Tim	Depres
Stanislav Popov	Epileps
Sergey Pavlov	Lung C
Olga Kuznetsova	Migrain
Oleg Sokolov	Allergy
Oksana Chernyshova	Varicos
Nikolay Fomin	Pancre
Nadezhda Saveleva	Tonsilli
Marina Sokolova	Cholec
Maria Petrova	Asthma
Maksim Nikiforov	Hyperte
Lidia Krylova	Diabete
Ivan Ivanov	Acute F
Ilya Borisov	Glauco
Elena Yakovleva	Anemia
Ekaterina Morozova	Osteoc
Dmitry Kravtsov	Chroni
Anna Sidorova	Pneum
Andrey Zuev	Stomac
Alyona Gavrilova	Obesity
Aleksei Smirnov	Gastriti

Дата приёма ▲	
01.10.2024	Ac
02.10.2024	W
03.10.2024	C
04.10.2024	Ac
05.10.2024	W
08.10.2024	Ac
09.10.2024	W
10.10.2024	Ac
11.10.2024	C
12.10.2024	Ac
13.10.2024	W
14.10.2024	Ac
15.10.2024	W
16.10.2024	Ac
17.10.2024	C
18.10.2024	Ac
19.10.2024	W
20.10.2024	Ac
21.10.2024	C
22.10.2024	Ac
23.10.2024	W
24.10.2024	Ac
25.10.2024	W
26.10.2024	Ac

Поиск по имени пациента/врача и названию болезни:

СохранитьДобавитьУдалить

Поиск по:Имени пациентаTimПоиск

Имя пациента	Болезнь	Врач	Специализация врача	Дата приёма ▲	Статус
Ivan Ivanov	Acute Respiratory Viral Infection	Doctor A	Therapist	01.10.2024	Accepted
Tim	Depression	Doctor B	Psychiatrist	02.10.2024	Waiting
Maria Petrova	Asthma	Doctor C	Therapist	03.10.2024	Cancelled
Aleksei Smirnov	Gastritis	Doctor D	Gastroenterologist	04.10.2024	Accepted

СохранитьДобавитьУдалить

Поиск по:Имени врачаВПоиск

Имя пациента	Болезнь	Врач	Специализация врача	Дата приёма ▲	Статус
Ivan Ivanov	Acute Respiratory Viral Infection	Doctor A	Therapist	01.10.2024	Accepted
Tim	Depression	Doctor B	Psychiatrist	02.10.2024	Waiting
Maria Petrova	Asthma	Doctor C	Therapist	03.10.2024	Cancelled
Aleksei Smirnov	Gastritis	Doctor D	Gastroenterologist	04.10.2024	Accepted

СохранитьДобавитьУдалить

Поиск по:Названию болезниDepressionПоиск

Имя пациента	Болезнь	Врач	Специализация врача	Дата приёма ▲	Статус
Ivan Ivanov	Acute Respiratory Viral Infection	Doctor A	Therapist	01.10.2024	Accepted
Tim	Depression	Doctor B	Psychiatrist	02.10.2024	Waiting
Maria Petrova	Asthma	Doctor C	Therapist	03.10.2024	Cancelled
Aleksei Smirnov	Gastritis	Doctor D	Gastroenterologist	04.10.2024	Accepted

Удаление выбранной строки:

СохранитьДобавитьУдалить 脳(のう)

Имя пациента	Болезнь
Иван Иванов	ОРВИ
ティム	幸せになりたい

СохранитьДобавитьУдалить 脳(のう)

Имя пациента	Болезнь
Иван Иванов	ОРВИ

Возможность редактирования:

Имя пациента	Болезнь	Врач	Специализация врача	Дата приёма	
Ivan Ivanov	Acute Respiratory Viral Infection	Doctor Ambus	Therapist	01.10.2024	Accepted
Tim	Depression	Doctor B	Psychiatrist	02.10.2024	Waiting

Удаление выбранной вкладки:

Файл

СохранитьДобавитьУдалить

Поиск по: Названию болезниDepressionПоиск

Имя пациента	Болезнь	Врач	Специализация врача	Дата приёма	Статус
Tim	Depression	Doctor B	Psychiatrist	02.10.2024	Waiting
Maria Petrova	Asthma	Doctor C	Therapist	03.10.2024	Accepted

Создание .jrxml файла (шаблона):

Data Adapter Wizard

DataAdapter File

Create a DataAdapter in a file

Enter or select the parent folder:  
MyReports

MyReports

File name: DataAdapter.xml

Next >

JasperReports DataSource Provider class

Microsoft Excel (XLS, XLSX)

Mondrian OLAP Connection

MongoDB Connection

Query Executor adapter

Random records

**XML document**

XML/A Server

Data Adapter Wizard

Data Adapter

XML document

Name: \_TestAdapter

File/URL: C:\Users\User\Desktop\LETT\OOOP\Java projects\Hospital-lab File

Enable namespaces support

Use the report Xpath expression when filling the report

Create data source using this expression :

Select Expression :

Data Source

Select a Data Source and design the query.

Data Adapter New Data Adapter - XML document

To enter the query you can write it in the related area or double-click on. Pressing next will try to automatically discover the fields.

patients

patient

patient

patient

/patients/patient

Fields

Please select dataset fields

Dataset Fields

name

disease

doctor

specialization

date

status

Fields

name

disease

doctor

specialization

date

status

name

\$F{name}

Сгенерированные шаблоны:

## ClinicAPP PDF format ^\_^

name	disease	doctor	specialization	date	status
Ivan Ivanov	Acute Respiratory Viral Infection	Doctor A	Therapist	01.10.2024	Accepted
Tim	Depression	Doctor B	Psychiatrist	02.10.2024	Waiting
Maria Petrova	Asthma	Doctor C	Therapist	03.10.2024	Canceled
Aleksei Smirnov	Gastritis	Doctor D	Gastroenterologist	04.10.2024	Accepted
Olga Kuznetsova	Migraine	Doctor E	Neurologist	05.10.2024	Waiting
Victor Tikhonov	Arthritis	Doctor F	Rheumatologist	08.10.2024	Accepted
Anna Sidorova	Pneumonia	Doctor G	Pulmonologist	09.10.2024	Waiting
Maksim Nikiforov	Hypertension	Doctor H	Cardiologist	10.10.2024	Accepted
Lidia Krylova	Diabetes	Doctor I	Endocrinologist	11.10.2024	Canceled
Sergey Pavlov	Lung Cancer	Doctor J	Oncologist	12.10.2024	Accepted
Ekaterina Morozova	Osteochondrosis	Doctor K	Orthopedist	13.10.2024	Waiting
Nikolay Fomin	Pancreatitis	Doctor L	Gastroenterologist	14.10.2024	Accepted
Vera Lebedeva	Insomnia	Doctor M	Neurologist	15.10.2024	Waiting
Oleg Sokolov	Allergy	Doctor N	Allergist	16.10.2024	Accepted
Nadezhda Saveleva	Tonsillitis	Doctor O	Otolaryngologist	17.10.2024	Canceled
Yuriy Grishin	Kidney Disease	Doctor P	Nephrologist	18.10.2024	Accepted
Marina Sokolova	Cholecystitis	Doctor Q	Surgeon	19.10.2024	Waiting
Ilya Borisov	Glaucoma	Doctor R	Ophthalmologist	20.10.2024	Accepted

# ClinicApp HTML

name	disease	doctor	specialization	date	status
Ivan Ivanov	Acute Respiratory Viral Infection	Doctor A	Therapist	01.10.2024	Accepted
Tim		Doctor B	Psychiatrist	02.10.2024	Waiting
Maria Petrova	Asthma	Doctor C	Therapist	03.10.2024	Canceled
Aleksei Smirnov	Gastritis	Doctor D	Gastroenterologist	04.10.2024	Accepted
Olga Kuznetsova	Migraine	Doctor E	Neurologist	05.10.2024	Waiting
Victor Tikhonov	Arthritis	Doctor F	Rheumatologist	08.10.2024	Accepted
Anna Sidorova	Pneumonia	Doctor G	Pulmonologist	09.10.2024	Waiting
Maksim Nikiforov	Hypertension	Doctor H	Cardiologist	10.10.2024	Accepted
Lidia Krylova	Diabetes	Doctor I	Endocrinologist	11.10.2024	Canceled
Sergey Pavlov	Lung Cancer	Doctor J	Oncologist	12.10.2024	Accepted
Ekaterina Morozova	Osteochondrosis	Doctor K	Orthopedist	13.10.2024	Waiting
Nikolay Fomin	Pancreatitis	Doctor L	Gastroenterologist	14.10.2024	Accepted
Vera Lebedeva	Insomnia	Doctor M	Neurologist	15.10.2024	Waiting
Oleg Sokolov	Allergy	Doctor N	Allergist	16.10.2024	Accepted
Nadezhda Saveleva	Tonsillitis	Doctor O	Otolaryngologist	17.10.2024	Canceled
Yuriy Grishin	Kidney Disease	Doctor P	Nephrologist	18.10.2024	Accepted
Marina Sokolova	Cholecystitis	Doctor Q	Surgeon	19.10.2024	Waiting
Ilya Borisov	Glaucoma	Doctor R	Ophthalmologist	20.10.2024	Accepted

**Разработка объектной модели приложения для управления данными клиники**

Объектная модель приложения не описывает внутреннюю структуру программного комплекса в коде, а отображает основные понятия предметной области клиники в виде совокупности типов объектов (сущностей). Эти сущности выделяются из анализа требований и прецедентов, связанных с учётом пациентов, проведением приёмов, работой врачей и назначением процедур.

---

## 1. Сущности

На диаграмме каждая сущность изображается прямоугольником, внутри которого указываются её **название**, **атрибуты** и **операции**. Ниже представлены основные сущности для рассматриваемой предметной области:

---

### Пациент (Patient)

- **Атрибуты:**
    - *имя*: String — имя пациента.
    - *болезнь*: String — заболевание, с которым обратился пациент.
    - *статус*: String — текущее состояние приёма (например, «Waiting», «Accepted», «Canceled»).
  - **Операции:**
    - `добавить()` — добавляет информацию о новом пациенте в систему.
    - `удалить()` — удаляет сведения о пациенте из списка.
    - `редактировать()` — изменяет данные о пациенте (например, обновляет болезнь или статус).
- 

### Врач (Doctor)

- **Атрибуты:**
    - *имя*: String — имя врача.
    - *специализация*: String — специализация (терапевт, хирург и т. д.).
  - **Операции:**
    - `добавить()` — добавляет нового врача в систему.
    - `удалить()` — удаляет врача из списка.
    - `редактировать()` — редактирует сведения о враче.
- 

### Приём (Appointment)

- **Атрибуты:**
    - *дата*: Date — дата приёма пациента.
    - *время*: Time — время начала приёма.
    - *врач*: String (или ссылка на объект Doctor) — врач, ведущий приём.
    - *пациент*: String (или ссылка на объект Patient) — пациент, записанный на приём.
  - **Операции:**
    - `добавить()` — создаёт новый приём (запись в расписании).
    - `удалить()` — отменяет приём.
    - `редактировать()` — корректирует время, дату или статус приёма.
-

## Пользователь (User)

- **Атрибуты:**
    - *имя*: String — логин пользователя (например, имя сотрудника).
    - *пароль*: String — пароль для доступа к системе.
  - **Операции:**
    - `регистрация()` — регистрирует нового пользователя (сотрудника) в системе.
    - `вход()` — авторизует пользователя и предоставляет доступ к функционалу.
    - `выход()` — завершает текущую пользовательскую сессию.
- 

## 2. Ассоциации между сущностями

Ассоциации описывают отношения между разными сущностями. Они обозначаются линиями на диаграмме, сопровождаются названием (семантикой) и указывают *кратность* (количество возможных связей).

- **Пациент — Приём**
    - Ассоциация: «записан на»
    - Кратность: 1 .. \* со стороны пациента (один пациент может иметь много приёмов) и 1 со стороны приёма (каждый приём связан конкретно с одним пациентом).
  - **Врач — Приём**
    - Ассоциация: «ведёт»
    - Кратность: 1 .. \* для врача (врач может вести несколько приёмов) и 1 со стороны приёма (каждый приём ведётся конкретным врачом).
  - **Пользователь — (Пациент, Врач, Приём)**
    - Пользователь (например, администратор или регистратор клиники) может управлять (добавлять, удалять, редактировать) данными о пациентах, врачах и приёмах.
- 

## 3. Элементы диаграммы сущностей

### 1. Сущности:

- **Пациент (Patient)**
  - *имя*: String
  - *болезнь*: String
  - *статус*: String
  - Операции: `добавить()`, `удалить()`, `редактировать()`
- **Врач (Doctor)**
  - *имя*: String
  - *специализация*: String
  - Операции: `добавить()`, `удалить()`, `редактировать()`
- **Приём (Appointment)**
  - *дата*: Date
  - *время*: Time
  - *врач*: String (или Doctor)
  - *пациент*: String (или Patient)

- Операции: `добавить()`, `удалить()`, `редактировать()`
- **Пользователь (User)**
  - *имя*: String
  - *пароль*: String
  - Операции: `регистрация()`, `вход()`, `выход()`

## 2. Ассоциации:

- **Пациент и Приём**: «записан на» (1 .. \* / 1)
- **Врач и Приём**: «ведёт» (1 .. \* / 1)
- **Пользователь и (Пациент, Врач, Приём)**: «управляет» (означает, что пользователь создаёт, редактирует или удаляет записи)

---

Таким образом, объектная модель приложения отражает структуру предметной области клиники, позволяя наглядно увидеть, как основные сущности (пациент, врач, приём) и пользовательские операции (добавление, редактирование, удаление, регистрация) связаны между собой. Эта модель служит базисом для дальнейшей реализации логики в коде, создания пользовательского интерфейса и взаимодействия с хранилищами данных (например, XML-файлами).

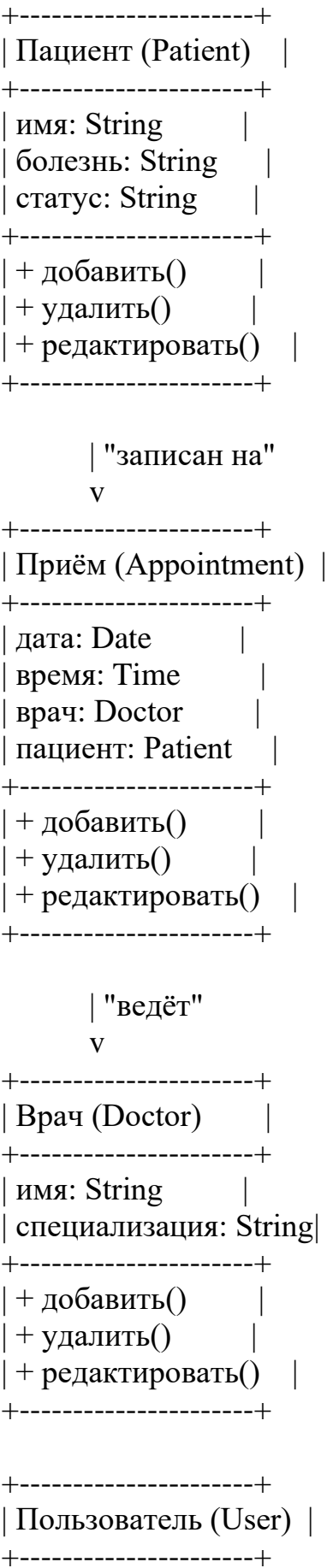
Ниже приведён **пример текстового представления** объектной модели для предметной области клиники (пациенты, врачи, приёмы, пользователи). Подобная диаграмма отражает сущности, их атрибуты и операции, а также показывает ассоциации («записан на», «ведёт» и т. д.). Обратите внимание, что это лишь одна из возможных реализаций в стиле ASCII; её можно дорабатывать или расширять при необходимости.

---



Текстовая диаграмма сущностей (пример)

Copy code



```

| имя: String      |
| пароль: String   |
+-----+
| + регистрация() |
| + вход()         |
| + выход()        |
+-----+

```

В данной диаграмме:

- **Пациент (Patient)** имеет атрибуты *имя*, *болезнь*, *статус* и операции для добавления, удаления, редактирования данных.
- **Приём (Appointment)** хранит *дату*, *время*, ссылки на *врача* и *пациента*.
- **Врач (Doctor)** характеризуется *именем* и *специализацией*.
- **Пользователь (User)** — сущность для управления доступом (авторизация, регистрация, завершение сеанса).

Ассоциации здесь показаны в виде пояснительных стрелок:

- Пациент «записан на» приём.
- Врач «ведёт» приём.
- Пользователь может «управлять» (или «создавать / редактировать / удалять») записями о пациентах, врачах и приёмах.

## Детальное описание операций (пример)

Ниже — пример, как можно детализировать некоторые операции в стиле «Входные параметры / Выходные параметры». Вы можете аналогично расписать все остальные методы.

### 1. Пациент (Patient)

- **Добавить пациента**
  - **Назначение:** Создаёт новую запись о пациенте в системе.
  - **Входные параметры:**
    - *имя*: String — имя пациента.
    - *болезнь*: String — заболевание.
    - *статус*: String — статус (например, «Waiting», «Accepted»).
  - **Выходные параметры:**
    - *успех*: boolean — флаг, указывающий на результат (успешно или нет).
- **Удалить пациента**
  - **Назначение:** Удаляет пациента из списка.
  - **Входные параметры:**
    - *идентификатор* или *индекс* (зависит от реализации)
  - **Выходные параметры:**
    - *успех*: boolean
- **Редактировать пациента**
  - **Назначение:** Обновляет данные о пациенте (новое имя, новая болезнь, новый статус и т. д.).
  - **Входные параметры:**
    - *идентификатор* (int) — уникальный ID пациента.

- *новоеИмя*: String (может быть пустым, если без изменения).
- *новаяБолезнь*: String
- *новыйСтатус*: String
- **Выходные параметры:**
  - *успех*: boolean

## 2. Приём (Appointment)

- **Добавить приём**
  - **Назначение:** Создаёт новую запись о приёме (расписание).
  - **Входные параметры:**
    - *дата*: Date
    - *время*: Time
    - *врач*: String (или Doctor)
    - *пациент*: String (или Patient)
  - **Выходные параметры:**
    - *успех*: boolean
- **Удалить приём**
  - **Назначение:** Удаляет приём из расписания.
  - **Входные параметры:**
    - *идентификатор* (int) — уникальный ID приёма.
  - **Выходные параметры:**
    - *успех*: boolean
- **Редактировать приём**
  - **Назначение:** Изменяет дату, время, врача или пациента в существующей записи.
  - **Входные параметры:**
    - *идентификатор* (int) — уникальный ID приёма.
    - *новаяДата*: Date
    - *новоеВремя*: Time
    - *новыйВрач*: String
    - *новыйПациент*: String
  - **Выходные параметры:**
    - *успех*: boolean

## 3. Врач (Doctor)

- **Добавить врача**
  - **Назначение:** Добавляет нового врача.
  - **Входные параметры:**
    - *имя*: String
    - *специализация*: String
  - **Выходные параметры:**
    - *успех*: boolean
- **Удалить врача**
  - **Назначение:** Удаляет врача из списка.
  - **Входные параметры:**
    - *идентификатор* (int)
  - **Выходные параметры:**

- *успех*: boolean
  - **Редактировать врача**
    - **Назначение:** Изменяет информацию о враче (имя, специализация).
    - **Входные параметры:**
      - *идентификатор* (int)
      - *новоеИмя*: String
      - *новаяСпециализация*: String
    - **Выходные параметры:**
      - *успех*: boolean
- #### 4. Пользователь (User)
- **Регистрация**
    - **Назначение:** Создаёт учётную запись пользователя в системе.
    - **Входные параметры:**
      - *имя*: String — логин.
      - *пароль*: String — пароль.
    - **Выходные параметры:**
      - *успех*: boolean
  - **Вход**
    - **Назначение:** Авторизует существующего пользователя.
    - **Входные параметры:**
      - *имя*: String
      - *пароль*: String
    - **Выходные параметры:**
      - *успех*: boolean
      - *сообщение*: String — «Успешный вход» или «Неверный пароль»
  - **Выход**
    - **Назначение:** Завершает сессию пользователя.
    - **Входные параметры:** (*нет*)
    - **Выходные параметры:**
      - *успех*: boolean

---

Таким образом, подобное **текстовое представление диаграммы сущностей** (сущности, их атрибуты, методы, а также возможные ассоциации) даёт общее понимание структуры предметной области и облегчает дальнейшую реализацию приложения.

## 1. Построение диаграммы классов для приложения управления клиникой

Диаграмма классов (*class diagram*) иллюстрирует будущие программные классы и интерфейсы на основе объектной модели клиники. Для каждого класса указываются три раздела: имя класса, состав его атрибутов и методы (операции). Графически класс отображается прямоугольником, в котором последовательно перечисляются название, поля и методы.

---

## 2. Структура классов

Ниже описаны основные классы, выделенные в системе управления клиникой.

---

### Пациент (Patient)

- **Атрибуты:**
    - + имя: String — имя пациента.
    - + болезнь: String — болезнь, с которой пациент обратился.
    - + статус: String — текущий статус приёма (например, «Waiting», «Accepted», «Canceled»).
  - **Методы:**
    - + добавить(): boolean — добавляет нового пациента.
    - + удалить(): boolean — удаляет пациента из списка.
    - + редактировать(): boolean — редактирует информацию о пациенте (имя, болезнь, статус).
- 

### Врач (Doctor)

- **Атрибуты:**
    - + имя: String — имя врача.
    - + специализация: String — специализация врача (терапевт, хирург и т. д.).
  - **Методы:**
    - + добавить(): boolean — добавляет нового врача.
    - + удалить(): boolean — удаляет врача из списка.
    - + редактировать(): boolean — редактирует информацию о враче (имя, специализация и т. д.).
- 

### Приём (Appointment)

- **Атрибуты:**
    - + дата: Date — дата приёма.
    - + время: Time — время начала приёма.
    - + врач: String — указание, какой врач ведёт приём.
    - + пациент: String — указание, какой пациент записан на приём.
  - **Методы:**
    - + добавить(): boolean — добавляет новую запись о приёме.
    - + удалить(): boolean — удаляет приём из расписания.
    - + редактировать(): boolean — редактирует данные о приёме (дату, время, врача, пациента).
-

## Пользователь (User)

- **Атрибуты:**
    - + имя: String — имя (логин) пользователя в системе.
    - + пароль: String — пароль для входа.
  - **Методы:**
    - + регистрация(): boolean — регистрирует нового пользователя в системе.
    - + вход(): boolean — даёт возможность пользователю войти под своим логином и паролем.
    - + выход(): boolean — завершает сессию пользователя.
- 

## 3. Связи между классами

На диаграмме классов обычно различают три основных типа отношений: **ассоциация, агрегация и наследование**.

### 1. Ассоциации

- **Пациент ↔ Приём: «записан на»**
  - Кратность: у пациента может быть несколько приёмов (1..\*), тогда как каждый конкретный приём ссылается на одного пациента (1..1).
- **Врач ↔ Приём: «ведёт»**
  - Кратность: один врач ведёт несколько приёмов (1..\*), но каждый приём закреплён за одним конкретным врачом (1..1).

### 2. Агрегация

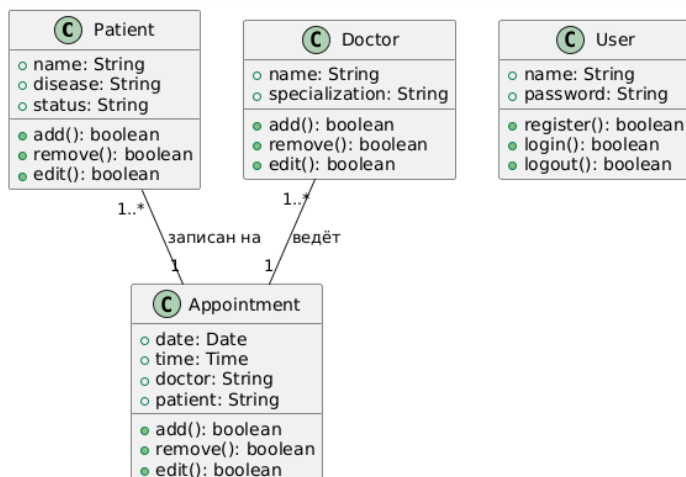
- Если, к примеру, класс «Врач» агрегирует класс «Приём» (или наоборот), то это может быть показано с помощью пустого ромба на линии связи. Это указывает, что, например, в рамках врача может вестись список нескольких приёмов.

### 3. Наследование

- Если в системе есть разные виды пользователей (например, «Администратор» и «Медсестра»), можно ввести классы Administrator и Nurse, которые будут наследовать базовый класс User. На диаграмме наследование обозначается стрелкой с белым треугольником, направленным от производного класса к базовому классу.

Таким образом, диаграмма классов даёт представление о том, **как** в приложении будут структурированы ключевые сущности (пациент, врач, приём, пользователь), **какие** у них есть данные (атрибуты) и **какие** операции они могут выполнять (методы), а также отражает **взаимосвязи** (ассоциации, агрегации, наследование) между классами.

Диаграмма классов:



## Описание поведения приложения для управления данными клиники (диаграмма последовательностей)

### 1. Общие сведения

Поведение приложения описывает, какие действия выполняются в ходе работы системы, не углубляясь при этом в детали механизмов реализации. Одним из способов наглядно представить поведение является **диаграмма последовательностей** (*sequence diagram*). Она показывает, в каком порядке происходят взаимодействия (запросы) между пользователем и объектами системы в рамках определённого сценария.

---

### 2. Идентификация пользователей и объектов

1. Определите, **какие пользователи** (например, администратор, оператор регистратуры) и **какие объекты** (классы «Пациент», «Врач», «Приём», «Пользователь» и т. д.) участвуют в рассматриваемом сценарии.
2. Изобразите эти объекты в верхней части диаграммы последовательностей в виде прямоугольников, подписав внутри прямоугольника (или под ним) имя объекта с подчёркиванием и название класса, которому он принадлежит (например, p1 : Patient).
3. Проведите от каждого объекта **вертикальную пунктирную линию**, обозначающую «линию жизни» (*lifeline*).

---

### 3. Выбор операций

- Из проектной документации (объектной модели, диаграммы классов) определите, какие **операции** выполняют данные объекты в данном сценарии (например, `добавитьПациента()`, `удалитьПациента()`, `редактироватьПриём()` и т. д.).
- Если каких-то операций, необходимых для сценария, раньше не описывали, добавьте их в модель.

---

### 4. Отображение запросов (сообщений)

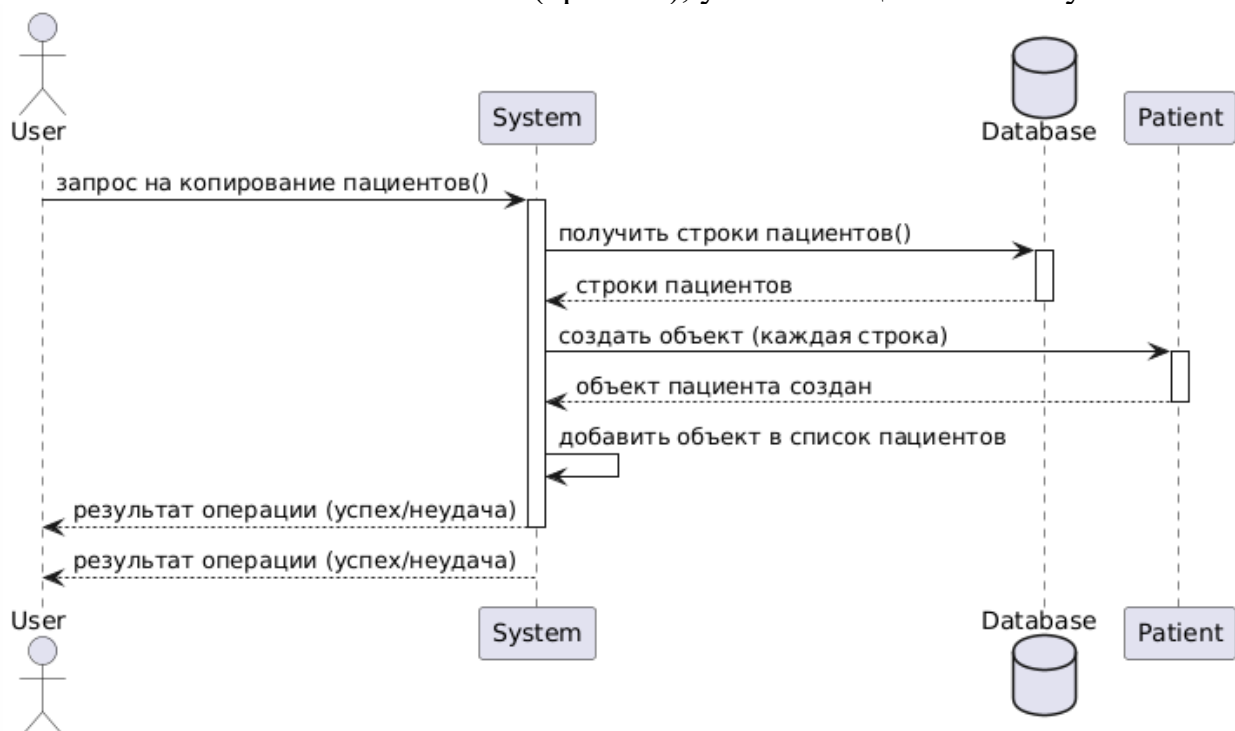
- Каждый запрос на выполнение операции отображается **горизонтальной стрелкой**: она начинается на линии жизни (lifeline) того объекта или пользователя, который инициирует вызов, и заканчивается на линии жизни объекта-исполнителя.
- Над стрелкой указывается **номер** запроса, **название** операции и при необходимости **параметры** (например, 1.1 addPatient(name, disease)).
- Расположение стрелки по вертикали показывает момент времени вызова (чем ниже, тем позже).

## 5. Порядок выполнения операций

- **Прямоугольник** на линии жизни объекта означает интервал выполнения конкретной операции (активность объекта).
- Для удобства используют **вложенную систему нумерации**: если операция 1 вызывает подоперации 1.1, 1.2 и т. д., то внутри 1.1 могут быть подшаги 1.1.1, 1.1.2 и так далее.
- Порядок стрелок сверху вниз отражает временную последовательность событий.

## 6. Условные ветвления и создание/уничтожение объектов

- При необходимости на диаграмме можно показать **логические условия** (if-else), когда одни операции выполняются только при выполнении определённого условия.
- Если объект **создаётся** в процессе сценария, его линия жизни начинается не сверху диаграммы, а с момента вызова конструктора (стрелка, указывающая на появление объекта).
- Если объект **уничтожается** до завершения сценария, его линия жизни заканчивается символом «X» (крестик), указывающим момент уничтожения.





## Построение диаграммы действий для приложения управления данными клиники

Диаграммы действий (*activity diagrams*) применяются для наглядного описания сложных операций, таких как добавление нового пациента, редактирование информации о приёмах или копирование данных из внешнего источника. Основная цель создания таких диаграмм — **визуализировать логику** (алгоритм) выполнения операций, определяя порядок действий и варианты переходов между ними.

**Графическое представление** диаграммы действий напоминает блок-схему:

- Вершины (*nodes*) соответствуют **действиям** (операциям системы).
- Рёбра (*edges*) показывают **переходы** от одного действия к другому.

Каждая диаграмма действий должна иметь **единственное начальное** и **единственное конечное** состояние. Обычно её строят вертикально, чтобы поток действий шёл сверху вниз.

---

### Основные элементы диаграммы действий

#### 1. Начальное состояние

Обозначает, с чего начинается процесс (например, «Запрос на добавление пациента»).

#### 2. Действия

Различные операции, такие как «Собрать данные о пациенте», «Проверить введённую информацию», «Внести пациента в базу данных» и т. д.

#### 3. Переходы

Связывают действия между собой, показывая **последовательность** их выполнения.

#### 4. Конечное состояние

Обозначает результат, например «Пациент успешно добавлен».

#### 5. Параллельные процессы (если нужны)

Диаграмма действий позволяет выделять и сливать **параллельные потоки** при помощи «штанги синхронизации» (линии, разделяющей и вновь объединяющей несколько ветвей выполнения).

---

### Пример диаграммы действий

Допустим, нам нужно отобразить **процесс добавления нового пациента** в клинику:

1. **Начальное состояние:** «Запрос на добавление пациента».
2. **Действие 1:** «Собрать данные пациента» (имя, болезнь, дата приёма и т. д.).
3. **Действие 2:** «Проверить корректность данных»:
  - Если данные корректны, переходим к следующему действию.
  - Если данные некорректны, переходим к действию «Вывести сообщение об ошибке» и завершаем процесс.
4. **Действие 3:** «Добавить пациента в базу данных» (или сохранить запись в XML).
5. **Конечное состояние:** «Пациент успешно добавлен».

Таким образом, диаграмма действий в формате «начальное состояние → последовательность действий → конечное состояние» помогает **наглядно** показать, какие шаги выполняются при добавлении пациента, какова их последовательность и

какие есть **разветвления** (условные переходы) в процессе.

Диаграмма:



## Руководство оператора

### 3.1 Назначение программы

Программа «Учёт пациентов» предназначена для автоматизации процессов, связанных с ведением информации о пациентах, врачах, расписании приёмов и статусах лечения в поликлинике (или другом медицинском учреждении). Она входит в состав автоматизированной системы учёта и администрирования данных и упрощает деятельность администраторов и сотрудников, отвечающих за хранение записей пациентов и их консультаций.

С помощью программы «Учёт пациентов» администратор может:

- Добавлять, редактировать и удалять информацию о пациентах.
- Добавлять, редактировать и удалять информацию о врачах.
- Добавлять, редактировать и удалять данные о приёмах (дате, времени, статусе).
- Получать справочную информацию о пациентах, врачах и расписании приёмов.

### 3.2 Условия выполнения программы

Программа работает в операционной среде Windows (версии XP, 7 и выше) и использует базу данных (например, MS Access или MySQL). Минимальные требования к компьютеру:

1. Процессор Pentium IV 1.5 ГГц или выше.
2. Не менее 2 Гб оперативной памяти.
3. Не менее 10 Гб свободного места на жёстком диске.
4. Видеокарта с объёмом памяти от 128 Мб.
5. Стандартная клавиатура.
6. Мышь или другой манипулятор.

### 3.3 Описание задачи

Приложение хранит сведения о пациентах, врачах и расписании приёмов.

Администратор имеет возможность добавлять, изменять и удалять эти данные. Также в системе можно управлять:

- Уровнями доступа пользователей (администратор, регистратор, врач и пр.).
- Информацией о пациентах и их назначениях.
- Датами и статусами приёмов (например, «Waiting», «Accepted», «Canceled»).
- Информацией о врачах и их специализации.

При написании программы на Java необходимо использовать:

- Открытые (public) и закрытые (private) члены классов.
- Наследование.
- Конструкторы с параметрами.
- Абстрактные базовые классы.
- Методы, подлежащие переопределению (виртуальные функции).
- Механизмы обработки исключений.
- Динамическое создание объектов.

На этапе проектирования была создана общая модель приложения, в которой определены основные сущности и связи между ними. В соответствии с этой моделью разработаны классы для хранения данных о пациентах, врачах и приёмах. Все данные хранятся в базе данных.

### 3.4 Входные и выходные данные

**Выходные данные** приложения — это таблицы, содержащие характеристики информационных объектов (например, список пациентов, расписание приёмов, сведения о врачах и их специализациях).

**Входные данные** (сведения о пациентах, врачах, времени и дате приёма) берутся из документации или вводятся вручную администратором в режиме диалога. Каждая характеристика (атрибут) может набираться вручную либо выбираться из предложенного списка значений.

---

## 1. Реализация отношений «многие ко многим»

В системе «Учёт пациентов» реализовано отношение «многие ко многим» между пациентами и врачами. Это означает, что один пациент может наблюдаться у нескольких врачей (например, у терапевта и кардиолога), а один врач может одновременно вести несколько пациентов. Подобная реализация повышает гибкость учёта и облегчает администрирование данных о приёмах.

## 2. Описание сущностей

### Пациенты (Patients)

- Один пациент может быть связан с несколькими врачами, если ему требуется консультация нескольких специалистов.
- Атрибуты пациента могут включать: `patient_id`, `name`, `age`, а также дополнительную информацию (например, `disease` или `status`).

### Врачи (Doctors)

- Один врач может наблюдать сразу нескольких пациентов.
- Атрибуты врача: `doctor_id`, `doc_name`, `specialty`.

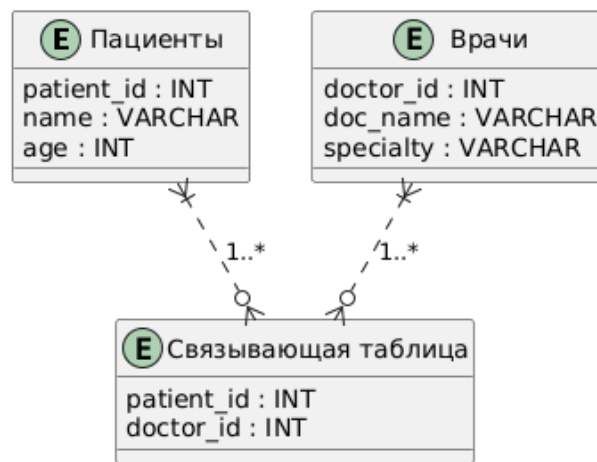
### Связующая таблица (Patient\_Doctor)

- Для реализации связи «многие ко многим» используется таблица `Patient_Doctor`, в которой хранятся идентификаторы пациента и врача.

- Атрибуты связующей таблицы: patient\_id, doctor\_id.

#### Описание схемы

- «Пациенты» может быть связана с несколькими записями в Patient\_Doctor, указывая на разных врачей, ведущих наблюдение.
- «Врачи» может быть связана с несколькими записями в Patient\_Doctor, отражая, что у врача есть несколько пациентов.
- Таблица Patient\_Doctor хранит пары (patient\_id, doctor\_id), описывая связь между конкретным пациентом и конкретным врачом. Это даёт возможность гибко изменять, кто у какого врача наблюдается и добавлять новые назначения при необходимости.



## Оценка качества пользовательского интерфейса

### Анализ удобства интерфейса

#### 1. Положительные стороны:

- **Простота:** интерфейс, основанный на таблицах (JTable) и кнопках (JButton), легко осваивается даже начинающим пользователям.
- **Минимализм:** в окне присутствуют только те элементы, которые действительно нужны для работы с пациентами и врачами.

#### 2. Возможные улучшения:

- **Подсказки (tooltips)** для кнопок, чтобы пользователь сразу понимал их назначение.
- **Более наглядная визуализация** (например, графики загрузки врачей или статистика приёмов).
- **Расширенный дизайн** с использованием JavaFX, чтобы интерфейс выглядел современнее и имел более гибкие элементы.

---

## Пользовательский интерфейс

Приложение содержит **интуитивно понятный GUI**, где основные операции с данными (пациентами, врачами, приёмами и т. д.) выполняются через кнопки и таблицы:

1. **Добавление новой строки** (например, пустой записи о пациенте или враче).
2. **Сохранение данных** во внешний файл (кнопка «Save XML»).
3. **Удаление выбранной строки** (кнопка «Delete»).
4. **Добавление данных** в таблицу (например, кнопка «Add»).
5. **Регистрация пользователя** (окно ввода логина/пароля перед входом).
6. **Экспорт данных** в форматы PDF и HTML для создания отчётов (например, по приёмам или статусам пациентов).
7. **Сброс фильтров и сортировок** (кнопка «Reset Filters»).
8. **Загрузка и сохранение XML** (кнопки «Load XML» и «Save XML»), позволяющие быстро восстанавливать или сохранять текущие записи.

Интерфейс упорядочен по вкладкам или панелям, что упрощает переключение между разными категориями данных (например, «Пациенты», «Врачи»).

---

## Перспективы развития

В дальнейшем приложение может быть доработано:

- **Внедрение статистических графиков** (например, для наглядного анализа расписания или численности пациентов).
- **Переход на JavaFX** для улучшения внешнего вида и интерактивности.
- **Автоматическая генерация отчётов** (в том числе графики для каждого врача или пациентов со схожими диагнозами).

---

## Описание структуры приложения

Приложение разработано на языке Java и разделено на **несколько логических компонентов**, что обеспечивает удобство сопровождения и масштабирования.

### 1. Графический интерфейс (GUI)

- **Библиотека Swing:** используется для создания окон, панелей, таблиц и

кнопок.

- **JTable и DefaultTableModel:**
  - JTable отображает списки пациентов, врачей и статусы приёмов.
  - DefaultTableModel хранит данные и предоставляет методы для вставки/удаления строк.
- **Кнопки (JButton):**
  - «Add» — добавляет новую запись (пациента или врача).
  - «Delete» — удаляет выделенную запись в таблице.
  - «Export to PDF/HTML» — экспорт текущей таблицы в указанный формат.
  - «Save XML / Load XML» — сохранение и загрузка данных в XML.
  - «Reset Filters» — сбрасывает фильтрацию или поиск.
- **Панели (JPanel):** группируют элементы интерфейса, разбивая программу на функциональные блоки (панель инструментов, область таблицы и т. д.).

## 2. Основные функции

- **Добавление, редактирование, удаление записей:**
  - Пользователь создаёт новую строку или выбирает существующую для правки/удаления.
- **Сохранение и загрузка данных в формате XML:**
  - Позволяет хранить информацию о пациентах, врачах и приёмах между запусками приложения.
- **Экспорт отчётов:**
  - Таблицы можно выгружать в PDF или HTML (например, для дальнейшей печати).
- **Фильтрация и сортировка:**
  - Быстрый поиск среди пациентов по имени, врачу или статусу приёма.

## 3. Логика управления данными

- **Работа с таблицами:**
  - Для каждого типа сущностей (пациенты, врачи и т. д.) есть свой DefaultTableModel. Добавление новой записи мгновенно отражается в JTable.
- **Работа с XML (класс XMLfile):**
  - Загрузка и сохранение списка записей в XML.
  - Структура XML согласуется со столбцами таблицы, что облегчает чтение/запись.
- **Экспорт (класс ReportGenerator):**
  - Генерирует PDF или HTML-отчёты на основе текущего состояния таблиц.
  - Происходит в отдельном потоке, чтобы интерфейс оставался отзывчивым.

## 4. Многопоточность

- Для длительных операций (экспорт, загрузка из XML) используется **несколько потоков** или ExecutorService.
- Это позволяет запускать тяжёлые задачи в фоне, не блокируя основную форму приложения.

## 5. Логирование

- Все действия пользователя (добавление, удаление, экспорт и т. п.) могут записываться в журнал через механизмы Logger.
- Логирование помогает анализировать проблемы и просматривать историю операций.

---

## Итог

Структура приложения опирается на чёткое разделение пользовательского интерфейса (Swing-компоненты) и бизнес-логики (управление таблицами, экспорт, работа с XML). Это облегчает дальнейшее развитие системы, добавление новых функций и поддержку многопоточности, а также делает интерфейс удобным и понятным для администратора или другого медицинского персонала.

## Проблемы, возникшие при разработке

### 1. Работа с XML:

- **Проблема:** Обработка некорректных данных в XML-файлах приводила к сбоям при загрузке.
- **Решение:** Внедрена дополнительная проверка структуры и содержимого XML перед загрузкой данных в таблицу.

### 2. Многопоточность:

- **Проблема:** Длительные операции, такие как экспорт данных в PDF или HTML, блокировали пользовательский интерфейс.
- **Решение:** Использование ExecutorService позволило выполнять фоновые задачи без блокировки GUI.

---

## Эффективность работы приложения

- **Производительность:**
    - Экспорт таблицы из 1000 записей в PDF выполняется менее чем за 2 секунды.
    - Загрузка XML-файла с 500 записями занимает около 1 секунды.
  - **Использование памяти:**
    - При работе с большими таблицами приложение потребляет около 50 МБ оперативной памяти.
-

## Заключение

В процессе разработки был создан **программный комплекс для управления списком пациентов клиники**, который соответствует заявленным требованиям и эффективно решает поставленные задачи. Приложение предоставляет удобный графический интерфейс для взаимодействия с данными о пациентах, врачах и расписании приёмов, а также обеспечивает автоматизацию рутинных операций.

### Основные достижения проекта:

#### 1. Функциональность:

- Реализованы операции добавления, редактирования и удаления данных о пациентах и врачах.
- Внедрены функции сортировки, поиска и фильтрации записей.
- Доступен экспорт данных в форматы **PDF** и **HTML**, что позволяет быстро создавать отчёты.

#### 2. Интуитивно понятный интерфейс:

- Элементы управления сгруппированы по логическим разделам.
- Кнопки и таблицы расположены удобно, что минимизирует время на освоение приложения.

#### 3. Оптимизация производительности:

- Благодаря **многопоточности** длительные задачи (например, экспорт отчётов) не блокируют интерфейс пользователя.
- Приложение остаётся отзывчивым даже при работе с большими объёмами данных.

---

## Преодолённые сложности

#### 1. Работа с XML:

- Обеспечена корректная обработка XML-документов с использованием `javax.xml`, что позволило избежать ошибок при загрузке и сохранении данных.

#### 2. Многопоточность:

- Внедрение **ExecutorService** позволило изолировать длительные операции и повысить стабильность интерфейса.

---

## Оценка достижений

- Программа успешно прошла тестирование на разных сценариях использования.
- Функциональность и производительность приложения соответствуют требованиям.
- Интерфейс является простым и интуитивно понятным, а система логирования обеспечивает прозрачность происходящих процессов.

---

## Перспективы развития

#### 1. Интеграция с базами данных:

- Подключение к полноценной базе данных (например, MySQL) для управления большими объёмами информации.

#### 2. Статистические инструменты:



- Добавление графиков и диаграмм для анализа данных о пациентах и врачах.
  - 3. **Обновление интерфейса:**
    - Переход на **JavaFX** для улучшения пользовательского опыта.
  - 4. **Аналитические отчёты:**
    - Автоматическое создание отчётов с диаграммами и ключевыми показателями.
  - 5. **Облачная синхронизация:**
    - Возможность синхронизации данных с облачным хранилищем для удалённого доступа.
- 

### **Заключительные выводы**

Разработанный **программный комплекс для управления пациентами клиники** успешно решает поставленные задачи, обеспечивая удобный интерфейс, стабильную работу и возможность дальнейшего расширения.

Проект позволил развить ключевые навыки в области:

- Проектирования объектно-ориентированных систем.
- Применения многопоточности и оптимизации производительности.
- Работы с XML-данными.

Полученные знания и опыт могут быть использованы при разработке более сложных и масштабных программных решений в будущем.

### **Ссылки:**

>> репозиторий:

[HTTPS://GITHUB.COM/ICONLTI/LTPROJECTS/TREE/MASTER/OOP/JAVA%20PROJECTS/COURSEWORK](https://github.com/iconlti/ltpjects/tree/master/oop/java%20projects/coursework)