|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |
| Институт кибербезопасности и цифровых технологий |
| Кафедра КБ-2 «Прикладные информационные технологии» |

**ПРАКТИЧЕСКАЯ РАБОТА №1 по дисциплине**

**«Технологии и методы программирования»**

***(наименование дисциплины)***

**Тема практической работы   
«Модельно-ориентированный подход к проектированию»**

Студент группы: БИСО-01-20 Митрофанов М.С.

*(шифр учебной группы) (Фамилия И.О)*

Преподаватель: Лесько С.А.

*(Фамилия И.О)*

Москва, 2022

**Порядок выполнения практической работы**

1. При помощи программы PlantUML либо любого редактора построить UML-диаграмму вариантов использования, диаграмму классов проектируемой информационной системы в соответствии с вариантом задания, а также диаграмму последовательности для наиболее часто используемых прецедентов. При построении диаграммы классов нужно добиться достаточной детализации информационной системы. Убедитесь в том, что использовали отношения dependency, aggregation/c¬omposition, generalization, описали размещение классов по пакетам проекта.
2. Подготовить отчет с включением диаграмм. Загрузить в GIT. Прикрепить ссылку

**Вариант 3**

**Информационная система поликлиники**

Обзор: информационная система поликлиники позволяет ставить и снимать больных с учета, записывать больных на прием к врачам, учитывать факт приема, а также позволяет вести историю болезни (медицинскую карту) больного.

**Диаграмма вариантов использования**

Данный код представляет собой диаграмму вариантов использования (use case diagram) для информационной системы поликлиники.

Заданные требования:

* ставить и снимать больных с учета
* записывать больных на прием к врачам
* учитывать факт приема
* вести историю болезни (медицинскую карту) больного.

Дополнительно можно указать, что система позволяет авторизованным пользователям (медицинским работникам и регистраторам) выполнять эти действия.

Используемые свойства UML:

Association: связь между «актерами» и вариантами использования.

Include: связь между вариантами использования. «Update Medical Record» включает в себя «Add Patient» и «Schedule Appointment».

UML код:

@startuml

left to right direction

actor Patient

actor Registrar

actor Doctor

actor Nurse

rectangle "Clinic System" {

usecase "Add Patient" as UC1

usecase "Remove Patient" as UC2

usecase "Schedule Appointment" as UC3

usecase "Update Medical Record" as UC4

usecase "View Medical Record" as UC5

usecase "Book Appointment" as UC6

Patient --> Registrar

Doctor --> UC3

Doctor --> UC4

Nurse --> UC3

Patient --> UC5

Registrar --> UC1

Registrar --> UC2

Registrar --> UC3

Registrar --> UC4

Registrar --> UC5

Patient --> UC6

Nurse --> Doctor : assists

UC6 ..> UC3 : include

}

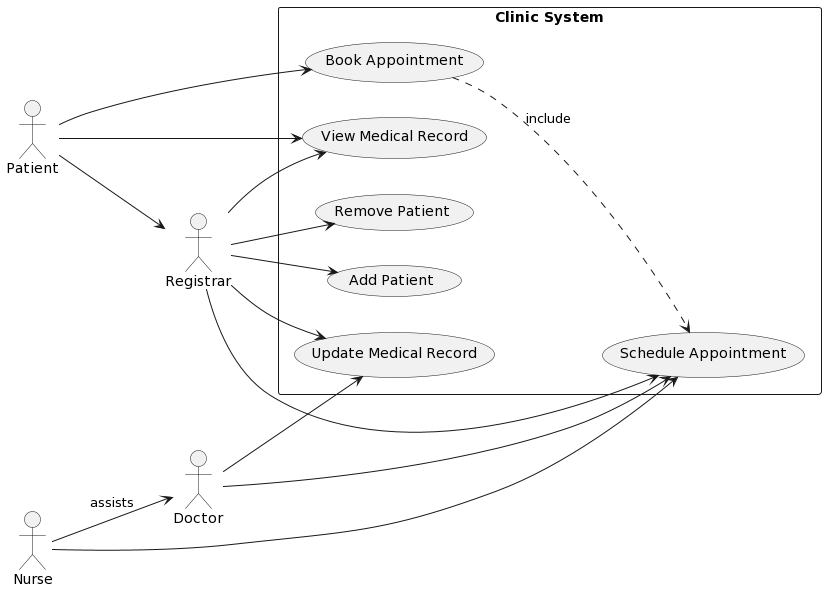
@enduml

Краткое описание

Пациент с помощью Регистратора может быть добавлен в систему, удален из системы поликлиники, записан на прием, также пациент может сам посмотреть записи в мед-книжке и записаться на прием ко врачу.

Врачи и Медсестры могут обновлять информацию в мед-книжке и записывать на прием.

Диаграмма вариантов использования системы на ранней стадии проектирования представлена на рисунке ниже.



**Диаграмма классов**

Данная диаграмма классов представляет систему клиники, которая включает в себя объекты, такие как пациенты, медицинский персонал, медицинские записи, а также регистратор, который управляет пациентами и записями.

В системе определены следующие классы:

ClinicSystem: класс, который представляет систему клиники. Он содержит объекты Registrar, Patient, Doctor, Nurse и MedicalRecord.

Person: абстрактный класс, который представляет человека, который работает в поликлинике, и его личную информацию имеет свойства: id, name, sex, address, birth date, marital status, phone. Класс Person нужен для абстрагирования общих свойств и методов, которые наследуются классами Doctor и Nurse. В данном случае, класс Person определяет общие свойства для любого человека, такие как идентификатор и имя, которые также будут доступны в классах Doctor и Nurse. Кроме того, класс Person определяет конструктор, который будет использоваться для создания объектов Doctor и Nurse, а также методы для получения идентификатора и имени.

Patient: класс пациента, в нем содержится вся информация о пациенте.

Doctor: класс, который наследуется от Person и представляет врача. Он имеет свойство specialty, которое указывает на его специализацию. Каждый врач имеет список записей, которые могут быть добавлены и удалены.

Nurse: класс, который наследуется от Person и представляет медсестру. Он имеет свойство specialty, которое указывает на ее специализацию. Каждая медсестра помогает доктору и может создать запись на прием.

MedicalRecord: класс, который представляет медицинскую запись, содержащую информацию о пациенте, докторе, диагнозе, назначении и дате записи.

Registrar: класс, который управляет пациентами и медицинскими записями. Он может зарегистрировать пациента, удалить пациента, назначить запись на прием, обновить медицинскую запись и получить медицинскую запись.

Используемые свойства UML:

Dependency: классы Doctor, Nurse, и Registrar зависят от класса Patient через методы scheduleAppointment() и updateMedicalRecord().

Generalization: класс Doctor и Nurse наследуются от абстрактного класса Person, который содержит общие данные о человеке.

Aggregation/Composition: классы Patient, Doctor, Nurse, MedicalRecord, и Appointment находятся внутри пакета Clinic System, что может означать, что они являются частями более крупной системы. Кроме того, классы Patient и Doctor содержат свойства типа "has", которые указывают на то, что объекты класса Patient и Doctor содержат или создают несколько объектов класса MedicalRecord и Appointment, соответственно.

UML код:

@startuml

package "Clinic System" {

abstract class Person {

-id: int

-name: string

-sex: string

-adress: string

-birth date: date

-marital status: string

-phone: string

+Person(id:int, name:string)

+getId():int

+getName():string

}

class Patient {

-id: int

-name: string

-sex: string

-adress: string

-birth date: date

-marital status: string

-phone: string

-medical insurance id: int

+Patient(id:int, name:string, address:string, phone:string, birth date:date,

marital status:string, medical insurance id:int, sex:string)

+scheduleAppointment(patient:Patient, doctor:Doctor, date:date): Appointment

+ viewMedicalRecord(): MedicalRecord

}

class Doctor extends Person {

-appointments: Appointment[\*]

-nurses: Nurse

-id: int

-name: string

- specialty: string

+ scheduleAppointment(patient: Patient, date: Date): Appointment

+ updateMedicalRecord(patient: Patient, medicalRecord: MedicalRecord): void

}

class Nurse extends Person {

-id: int

-name: string

- specialty: string

+ scheduleAppointment(patient: Patient, date: Date): Appointment

+ assistToDoctor(doctor: Doctor): Doctor

}

class Appointment {

- doctor: Doctor

- patient: Patient

- nurses: Nurse

- date: Date

- attended: bool

+ setAttended(attended: bool): void

}

class MedicalRecord {

-patient: Patient

-doctor: Doctor

-diagnosis: string

-prescription: string

-date: date

+MedicalRecord(patient:Patient, doctor:Doctor, diagnosis:string, prescription:string, date:date)

+getPatient():Patient

+getDoctor():Doctor

+getDiagnosis():string

+getPrescription():string

+getDate():date

}

class Registrar {

+registerPatient(name:string, address:string, phone:string, birth date:date,

martial status:string, medical insurance id:int, sex:string): Patient

+removePatient(patient:Patient): void

+scheduleAppointment(patient:Patient, doctor:Doctor, date:date): Appointment

+updateMedicalRecord(medicalRecord:MedicalRecord, diagnosis:string, prescription:string, date:date): void

+getMedicalRecord(patient:Patient): MedicalRecord

}

class ClinicSystem {

-registrar: Registrar

-patients: Patient[\*]

-doctors: Doctor[\*]

-nurses: Nurse[\*]

-medicalRecords: MedicalRecord[\*]

+addPatient(patient:Patient): void

+addDoctor(doctor:Doctor): void

+addNurse(nurse:Nurse): void

+getRegistrar(): Registrar

}

Patient "1" \*-- "0..\*" MedicalRecord : has

Doctor "1" \*-- "0..\*" MedicalRecord : creates

Doctor "1" \*-- "0..\*" Appointment : creates

Nurse "1" \*-- "0..\*" Appointment : creates

Registrar -> Patient : manages

Registrar -> MedicalRecord : manages

ClinicSystem \*-down-> Registrar : contains

ClinicSystem \*-down-> Patient : contains

ClinicSystem \*-down-> Doctor : contains

ClinicSystem \*-down-> Nurse : contains

Nurse --> Doctor : assists

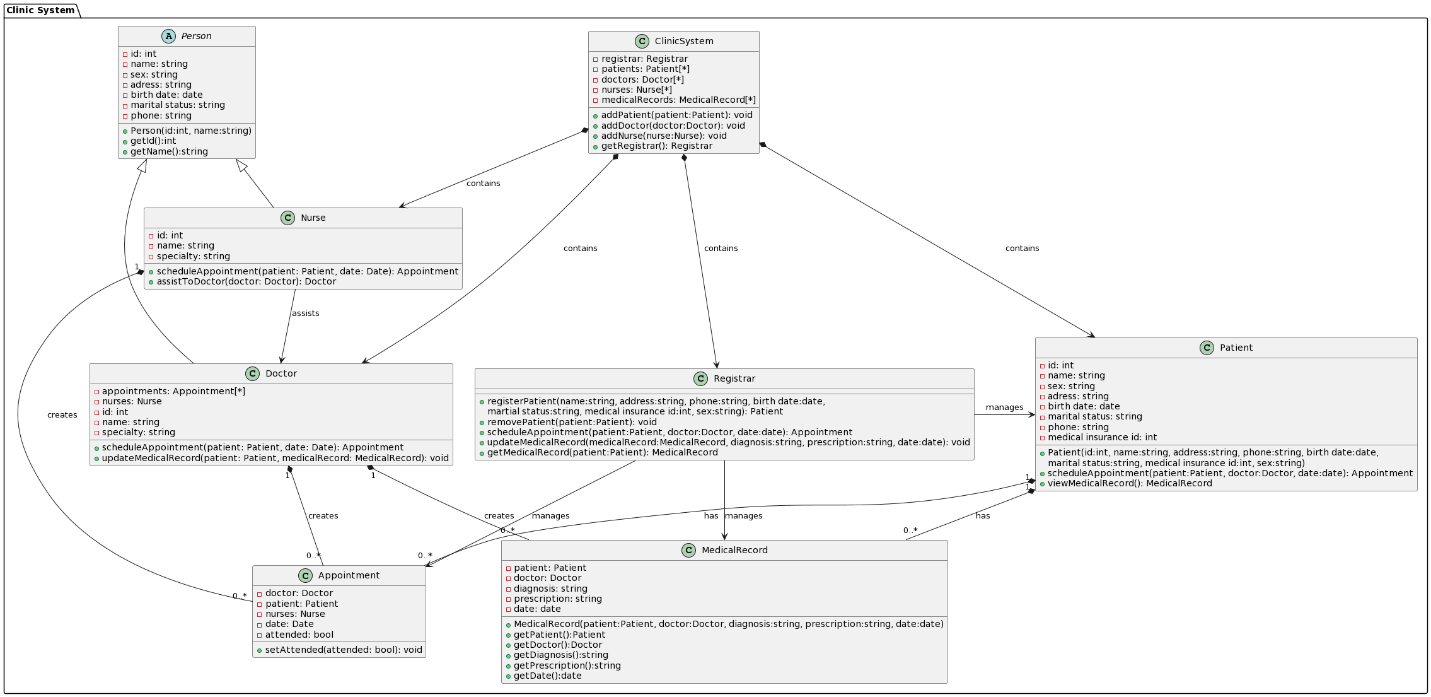
Patient "1" \*-- "0..\*" Appointment : has

Registrar -> Appointment : manages

}

@enduml

Диаграмма классов поликлиники:



**Диаграмма последовательности**

Описание

Сначала пациент регистрируется в системе путем отправки запроса на регистрацию в клинику через интерфейс пациента. Затем информация о пациенте передается регистратору, который создает запись о пациенте в системе и создает медицинскую карту.

После этого пациент записывается на прием к врачу через интерфейс пациента, и информация о записи передается регистратору. Регистратор создает запись о приеме и связывает ее с медицинской картой.

Затем врач заполняет медицинскую карту пациента, добавляя диагнозы и рецепты. После этого врач отмечает факт приема в системе, а медсестра помогает при осмотре пациента.

На диаграмме также показаны связи между участниками: пациент взаимодействует с клинической системой и регистратором, врачи заполняют медицинские карты, регистратор управляет записями на прием и медицинскими картами, а клиническая система содержит всех участников и регистратора.

Данный код UML создает диаграмму последовательности, которая описывает основные прецеденты информационной системы поликлиники:

@startuml

actor Patient

actor Doctor

actor Nurse

participant Registrar

participant ClinicSystem

participant MedicalRecord

Patient -> ClinicSystem: регистрация пациента

ClinicSystem -> Registrar: регистрация пациента

Registrar -> Patient: создание записи о пациенте

Patient -> ClinicSystem: запись на прием

ClinicSystem -> Registrar: запись на прием

Registrar -> MedicalRecord: создание медицинской карты

Doctor -> MedicalRecord: заполнение медицинской карты

Doctor -> MedicalRecord: изменение медицинской карты

Doctor -> Patient: осмотр пациента

Nurse -> Patient: помощь при осмотре

Doctor -> MedicalRecord: добавление диагноза и рецепта

Doctor -> ClinicSystem: отметка о приеме

Nurse -> ClinicSystem: отметка о приеме

@enduml

Диаграмма последовательности:

