УДК 004.6

Поварницын Е.Н., студент

4 курс, факультет «Информационные системы и технологии»

Северный Арктический Федеральный Университет

Россия, г. Архангельск

Povarnitsyn E. N., student

4rd year, faculty of Information systems and technologies»

Northern Arctic Federal University

Russia, Arkhangelsk

АВТОМАТИЗАЦИЯ ДЕЯТЕЛЬНОСТИ ПОЛИКЛИНИКИ НА ОСНОВЕ ОБЪЕКТНО- ОРИЕНТИРОВАННОГО ПОДХОДА

Automation of polyclinic activities based on an object-oriented approach

Аннотация:

Статья посвящается рассмотрению предметной области поликлиники. А также разработка программы по данной области на языке программирования Python.

Ключевые слова: Python, предметная область, поликлиника, программирование.

Annotation:

The article is devoted to the subject area of polyclinics. As well as developing a program in this area in the Python programming language.

Keyword:

Python, subject area, clinic, programming.

1 ПРЕДМЕТНАЯ ОБЛАСТЬ

1.1 Вербальное описание

Рассмотрим предметную область поликлиники. Современная поликлиника является крупным многопрофильным, специализированным лечебно-профилактическим учреждением, предназначенным оказывать медицинскую помощь и осуществлять комплекс профилактических мероприятий по оздоровлению населения и предупреждению заболеваний.

В ее функции входят:

а) оказание первой медицинской помощи при острых и внезапных заболеваниях, травмах;

б) лечение больных при обращении в поликлинику и на дому;

в) организация и проведение диспансеризации;

г) экспертиза временной нетрудоспособности;

д) освобождение больных от работы;

е) направление на врачебно-трудовую экспертную комиссию лиц с признаками стойкой утраты трудоспособности;

ж) направление больных на санаторно-курортное лечение;

з) своевременная госпитализация нуждающихся в стационарном лечении.

Поликлинику могут посещать пациенты, записавшиеся заранее в регистратуре. Пациенты приходят в указанное время и врачу, которому им нужно. Данные о пациенте находятся в их карточках, которые лежат в регистратуре. В них находятся имя, фамилия, отчество пациента, его дата рождения, социальный статус, регистрация, данные паспорта. А также посещения врачей, история болезней, учёт диспансеризации, амбулаторное лучение, срок нетрудоспособности пациента.

На приеме у врача пациент должен объяснить цель своего визита. Тем временем врач должен описать самочувствие больного, проконсультировать

и выписать должное лечение. После чего доктор должен назначить консультацию на другой день, если она требуется.

Информация на каждого врача, а именно имя, фамилия отчество, дата рождения, специальность, квалификация и должность расположена в архиве поликлиники.

1.2 Концептуальное описание

В данном сервисе персонал регистратуры будет заполнять карточку визитов пациентов к определённым врачам, так же заполнять саму карточку пациента и информацию о враче. Объектами системы будет врач и пациент, специальность врача, болезни, визит. Свойства объектов приведены в таблице 1.

Таблица 1 – Свойства объектов

Наименование объекта	Свойства объекта
Пациент	Имя пациента
	Фамилия пациента
	Возраст пациента
	Лечащий врач
Врач	Имя врача
	Фамилия врача
	Специальность врача
Болезнь	Наименование - Гастрит - Язва - Порок сердца - Сотрясение мозга - Простуда - Коронавирус - Ухудшение зрения - Потеря уха - Болезней нет
Специальность врача	Наименование - Глав врач - Кардиолог - Гастроэнтеролог - Терапевт - Окулист - Лор
Визит	Пациент
	Врач
	Болезнь

Определяем ограничения:

- при заполнении визита может быть выбран тот пациент, врач или болезнь, которые находятся уже в базе;

- при заполнении врача может быть выбрана та специальность, которая находятся уже в базе;

- действия добавления, чтения, редактирования и удаления может происходить если есть соединение с базой.

Схема взаимодействий представлена на рисунке 1.

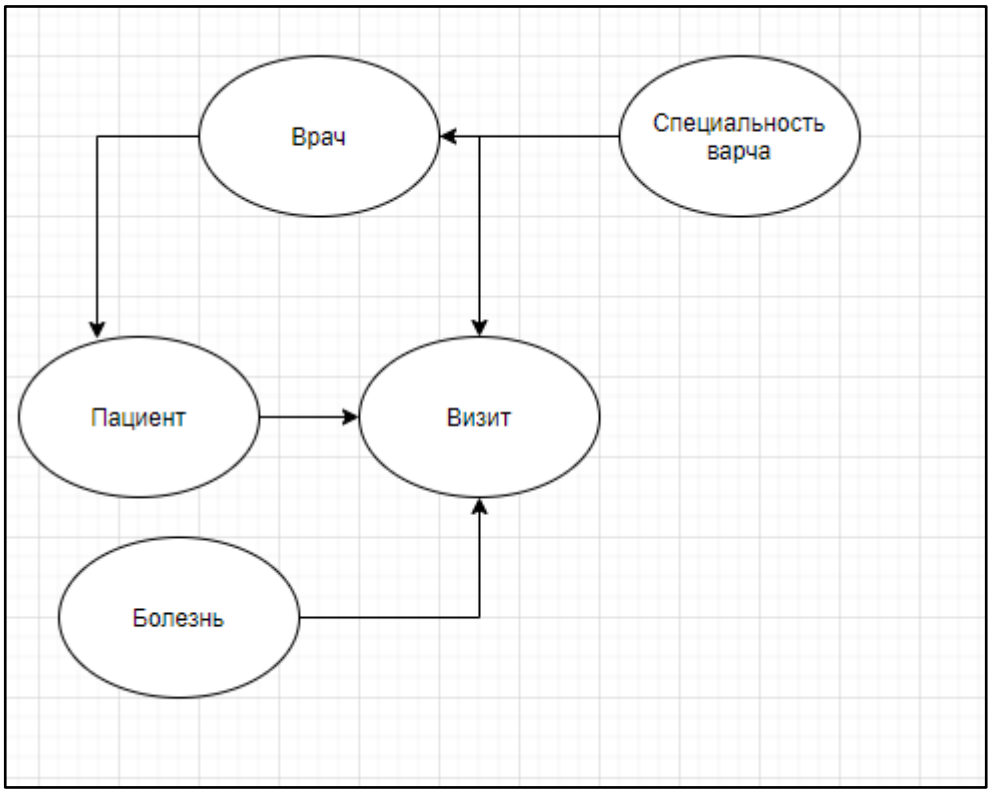


Рисунок 1 – Взаимодействие объектов

Сначала заполняется специальность врача, после чего, как специальность есть в базе, заполняется врач. Далее можно заполнить данные пациента, так как у пациента всегда нужно указывать его постоянного лечащего врача. Заполнение болезни может происходить в любой момент, но до начала заполнения визита. После этого в базе будет достаточно данных для заполнения визита к врачу.

2 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

Для примера будет взята реализация страницы визита пациентов к врачам. Для этого создаём ру файл. Здесь необходимо реализовать для начала счётчик ИД. Он будет выглядеть в соответствии с рисунком 2.

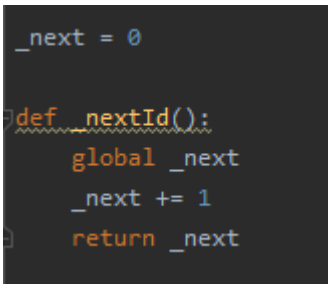
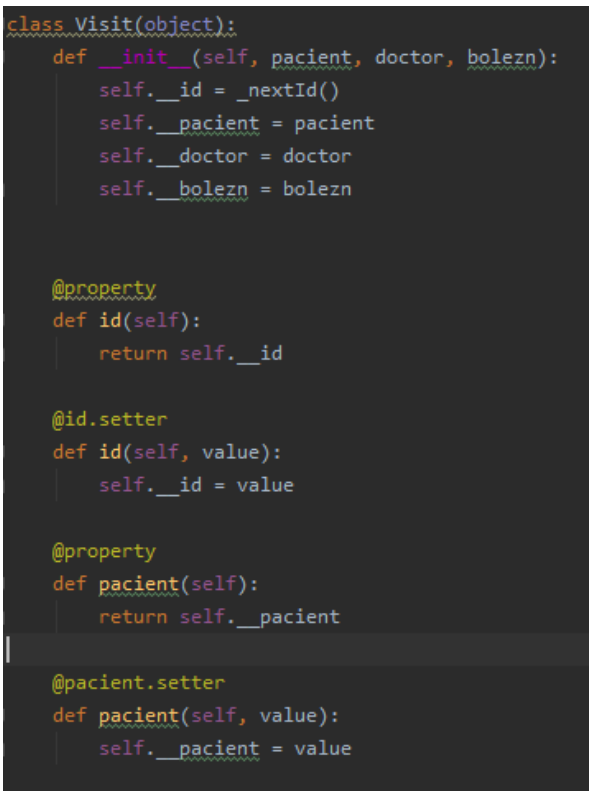


Рисунок 2 – Счётчик ИД

Здесь будет реализоваться глобальная переменная _next и метод, который будет прибавлять +1 при каждом его вызове.

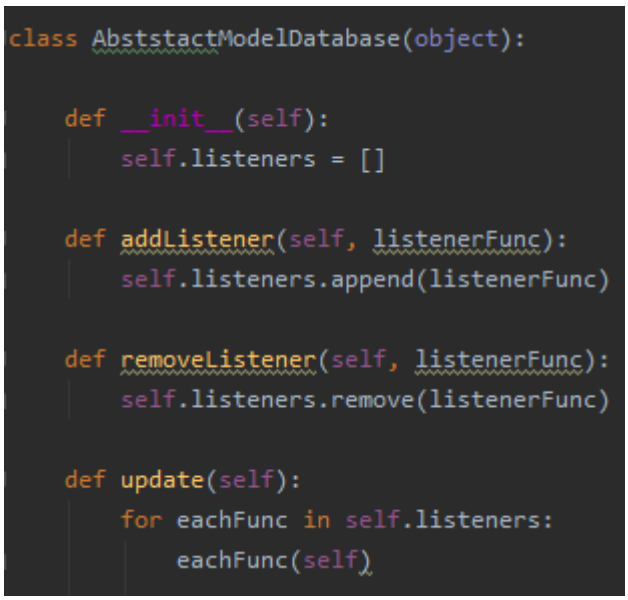
Далее рассмотрим класс Visit, данный класс будет хранить объекты, такие как ИД посещения, пациента, врача и болезнь. Данные объекты создадим приватными, а также добавим сеттеры для данных объектов и setter'ы, которые позволят обращаться к объектам как к функциям. Для Id необходимы сеттеры для последующего тестирования методов в системе. Выглядеть это будет в соответствии с рисунком 3.



```
class Visit(object):
    def __init__(self, pacient, doctor, bolezn):
        self.__id = _nextId()
        self.__pacient = pacient
        self.__doctor = doctor
        self.__bolezn = bolezn


    @property
    def id(self):
        return self.__id

    @id.setter
    def id(self, value):
        self.__id = value

    @property
    def pacient(self):
        return self.__pacient

    @pacient.setter
    def pacient(self, value):
        self.__pacient = value
```

Рисунок 3 – Класс Visit

Для последующие работы с базой данных следует добавить абстрактный класс, благодаря которому будут возможны различные действия. Выглядеть класс будет в соответствии с рисунком 4.



```
class AbststactModelDatabase(object):

    def __init__(self):
        self.listeners = []

    def addListener(self, listenerFunc):
        self.listeners.append(listenerFunc)

    def removeListener(self, listenerFunc):
        self.listeners.remove(listenerFunc)

    def update(self):
        for eachFunc in self.listeners:
            eachFunc(self)
```

Рисунок 4 – Абстрактный класс

Далее можно создавать класс, который будет взаимодействовать с базой, методе __init__ следует прописать ссылку на класс, индекс, равный 0, что позволит

считывать данные с самого начала, пути к файлам базы, и объекты, в которых будет храниться информация, считываемая с базы, выглядеть код будет в соответствии с рисунком 5.

```
def __init__(self):
    super(VisitDB, self).__init__()
    self.index = 0
    self.file = '\\VisitDB.pkl'
    self.doctorPath = '\\DoctorDB.pkl'
    self.pacientPath = '\\PacientDB.pkl'
    self.bolezn Path = '\\BoleznDB.pkl'
    self.mainDirectory = Path(__file__).parent.parent
    self.file = str(self.mainDirectory) + self.file
    self.doctorPath = str(self.mainDirectory) + self.doctorPath
    self.pacientPath = str(self.mainDirectory) + self.pacientPath
    self.bolezn Path = str(self.mainDirectory) + self.bolezn Path
    self.db = {}
    self.doctors = {}
    self.pacients = {}
    self.bolezns = {}
```

Рисунок 5 – Обозначение базы

После чего необходимо создать методы, позволяющие считывать и сохранять данные, в соответствии с рисунком 6.

```
def open_db(self):
    with open(self.file, 'rb') as file:
        self.db = pickle.load(file)

def open_doctor_db(self):
    with open(self.doctorPath, 'rb') as file:
        self.doctors = pickle.load(file)

def open_pacients_db(self):
    with open(self.pacientPath, 'rb') as file:
        self.pacients = pickle.load(file)

def open_bolezns_db(self):
    with open(self.boleznPath, 'rb') as file:
        self.bolezns = pickle.load(file)

def save_db(self):
    with open(self.file, 'wb') as file:
        pickle.dump(self.db, file)
```

Рисунок 6 – Чтение и запись данных

После чего добавим методы, которые будут срабатывать при нажатии на кнопки в самой программе. Первый метод позволит добавлять данные в базу. В него будут передаваться объект пациента, врача и болезнь. Далее будет автоматически приписываться ИД, обновляться данные и после чего сохраняться в базе. Код будет выглядеть в соответствии с рисунком 7.

```
def add(self, pacient, doctor,bolezn):
    visit = Visit(pacient, doctor, bolezn)
    if visit.id in self.db:
        visit.id = list(self.db.keys())[-1] + 1
    self.db[visit.id] = visit
    self.index = list(self.db.keys())[-1]
    self.update()
    self.save_db()
```

Рисунок 7 – Добавление данных

Кнопки далее и назад будут иметь схожий код, поэтому рассмотрим их обобщенно. Если данных методах будет происходить ошибка, то выйдет окно о том, что эта страница либо последняя, либо первоначальная. Если же массив

индекса будет равен ключу данных базы, то данные поменяются на следующие или же возвратятся назад. Код кнопки next будет выглядеть в соответствии с рисунком 8.

```
def next(self):
    if self.index == list(self.db.keys())[-1]:
        raise Exception("Последняя страница")
    else:
        try:
            self.index = self.index + 1
            while self.index <= list(self.db.keys())[-1] and self.index not in self.db.keys():
                self.index = self.index + 1
            self.update()
            return self.db[self.index]
        except Exception:
            raise Exception("Последняя страница")
```

Рисунок 8 – Метод следующей страницы

Далее рассмотрим метод удаления, в нём просто передаётся ИД посещения и через него удаляются данные, код выглядит в соответствии с рисунком 9.

```
def delete(self, id):
    del self.db[id]
    self.save_db()
    self.index = list(self.db.keys())[0]
    self.update()
```

Рисунок 9 - Метод удаления

Так же рассмотрим методы комбо боксов, в данных методах просто идёт рассмотрение базы с данными либо врача, либо пациента или болезни. Код будет выглядеть в соответствии с рисунком 10.

```
def fillComboboxDoctor(self):
    self.open_doctor_db()
    return self.doctors
```

Рисунок 10 – Метод комбо бокса

Полный код модели будет показан в приложении А.

Далее рассмотрим вид раздела посещений. В нём нужно объявить какая кнопка к какому методу относится, далее нужно прописать эти методы, ссылаясь

на тот код, который был разобран в модели раздела посещений. Код с присвоением команд и кнопок будет выглядеть в соответствии с рисунком 11.

```
def __init__(self, ui, Model):
    self.ui = ui
    self.model = Model
    self.model.addListener(self.update)

    self.ui.bAddVisit.clicked.connect(self.add)
    self.ui.bNextVisit.clicked.connect(self.next)
    self.ui.bPrevVisit.clicked.connect(self.prev)
    self.ui.bDeleteVisit.clicked.connect(self.delete)
    self.ui.bUpdateVisit.clicked.connect(self.edit)
    self.ui.cbVisitPacient.activated.connect(self.selectPacient)
    self.ui.cbVisitDoctor.activated.connect(self.selectDoctor)
    self.ui.cbVisitBolezn.activated.connect(self.selectBolezn)
```

Рисунок 11 – Присвоение кнопок и методов

Так же необходимо сделать окно вызвав программы, которая будет использовать и модель, и вид. Код будет выглядеть в соответствии с рисунком 12.

```
def __init__(self):
    super(MyWindow, self).__init__()
    self.ui = Ui_MainWindow()
    self.ui.setupUi(self)

    self.Doctor = DoctorDB()
    self.Pacient = PacientDB()
    self.Bolezn = BoleznDB()
    self.Special = SpecialDB()
    self.Visit = VisitDB()

    self.DoctorView = DoctorView(self.ui, self.Doctor)
    self.PacientView = PacientView(self.ui, self.Pacient)
    self.BleznView = BoleznView(self.ui, self.Bolezn)
    self.SpecialView = SpecialView(self.ui, self.Special)
    self.VisitView = VisitView(self.ui, self.Visit)

    self.ui.tabWidget.blockSignals(False)
    self.ui.tabWidget.currentChanged.connect(self.refresh)
```

Рисунок 12 – Запуск приложения

Таким образом, если соблюдать все шаги разработки, то получиться полноценная программа с чтением, записью, удалением и редактированием данных. Окно посещений будет выглядеть в соответствии с рисунком 13.

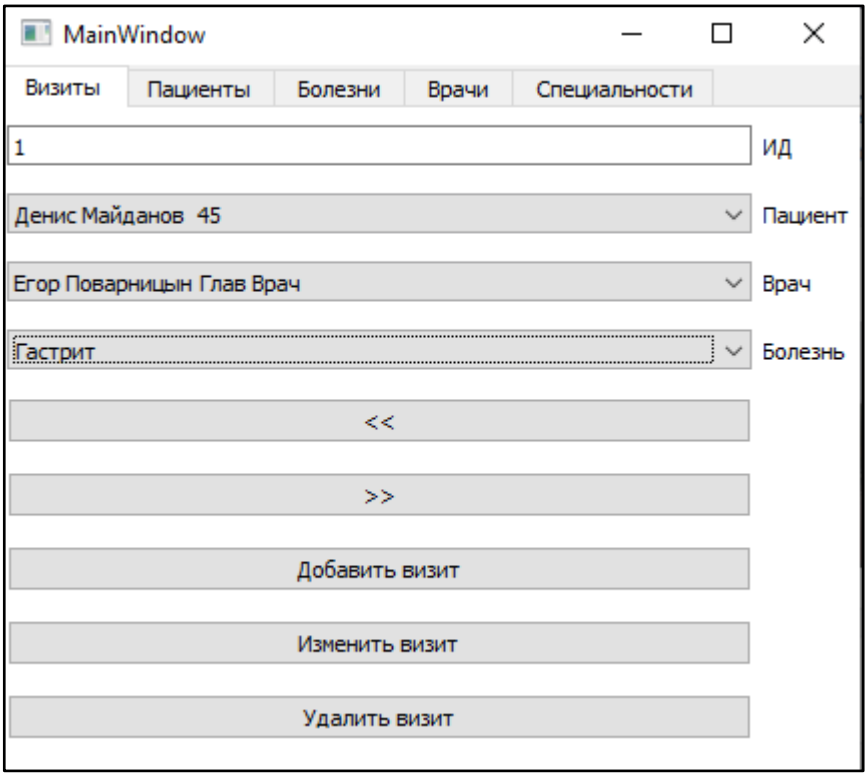


Рисунок 13 – Окно приложения

3 ТЕСТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ

Для тестирования программы будет использоваться unit тестирование. Возьмём опять же для примера тестирование раздела посещений пациента.

Юнит-тестирование кода является неотъемлемой частью жизненного цикла разработки программного обеспечения. Юнит-тесты также формируют основу для проведения регрессионного тестирования, то есть они гарантируют, что система будет вести себя согласно сценарию, когда добавятся новые функциональные возможности или изменятся существующие.

Проверим смогут ли принимать и получать данные программа, для этого пропишем код для принятия и получения данных, в соответствии с рисунком 14.

```
def testGetPacient(self):
    test_value = 'pacient'
    received_value = self.visit.pacient
    self.assertEqual(test_value, received_value)

def testSetPacient(self):
    test_value = 'new pacient'
    received_value = 'new pacient'
    self.visit.pacient = test_value
    self.assertEqual(test_value, received_value)
```

Рисунок 14 – Unit test

Создадим тесты для всех переменных в данном разделе, это ИД, доктор и болезнь, тест для пациента уже имеется.

Если всё было составлено правильно, то в результате при запуске получаем положительный результат в соответствии с рисунком 15.

```
Ran 8 tests in 0.004s

OK
```

Рисунок 15 – Результат тестирования

Так же были произведены тесты методов. Брался метод добавления и удаления данных. Код выглядит в соответствии с рисунком 16.

```
def testAddVisit(self):
    self.visit.add("Пациент", "Доктор", "Болезнь")
    pacient = "Пациент"
    doctor = "Доктор"
    bolezn = "Болезнь"
    id = len(self.visit.db)
    self.assertEqual(pacient, self.visit.db[id].pacient)
    self.assertEqual(doctor, self.visit.db[id].doctor)
    self.assertEqual(bolezn, self.visit.db[id].bolezn)
    self.visit.delete(id)

def testDeleteVisit(self):
    idVisit = list(self.visit.db.keys())[-1]
    self.visit.add('Пациент', 'Доктор', 'Болезнь')
    idDelVisit = list(self.visit.db.keys())[-1]
    self.visit.delete(idDelVisit)
    self.assertEqual(idVisit, list(self.visit.db.keys())[-1])
    self.visit.delete(idVisit)
```

Рисунок 16 – Тестирование методов

Здесь идёт добавление в базу данных, а после чего сравнение, добавлюсь ли запись в базу или нет. Так же есть метод тестирования удаления в нём так же создаётся запись, после чего сравнивается то, что такой записи нет в системе.