|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | http://mai.ru/life/brand/mai.gif | 1. **МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ** 2. **(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)** | |
| Институт №3 «Системы управления, информатика и электроэнергетика» |
| Кафедра 307 «Цифровые технологии и информационные системы» |

|  |
| --- |
| Лабораторная работа №4  **«Создание UML-диаграммы классов к программе курсового проекта»**  По дисциплине:  **«Методы и средства проектирования информационных систем и технологий»** |
|  |
|  |
|  |

Выполнили:

студенты группы М3О-412Б-18

Акимов В.Н.

Усенко Д.Е.

Проверил:

Максимов Н.А.

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

дата, подпись

Москва2021

# Цель работы

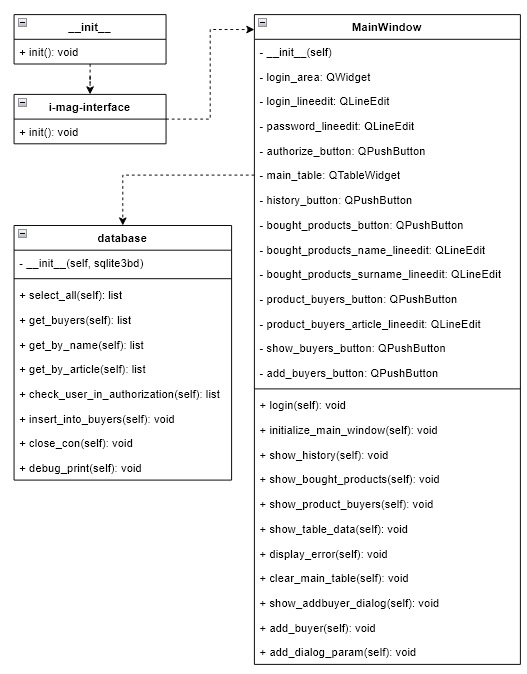
Составить UML –диаграмму классов для программы, сделанной к курсовому проекту. Листинг файлов программы представлен в приложении.

# Практическая часть

Программа состоит из 3 модулей:

* \_\_init\_\_.py – основной файл инициализации программы. Содержит функцию запуска окна программы.
* i\_mag\_interface.py – содержит класс MainWindow и функции для его инициализации.
* database.py – содержит функции для взаимодействия с базой данных.

В соответствии с модулями составим UML- диаграмму классов. Пунктирными стрелками обозначим зависимости классов. Полученная диаграмма классов представлена на рисунке 1.



*Рисунок 1 – Диаграмма классов.*

# Вывод

В ходе лабораторной работы была проанализирована структура программы и составлена UML-диаграмма классов.

# Приложение

Модуль \_\_init\_\_.py:

import Program.i\_mag\_interface as interface

interface.init()

Модуль i\_mag\_interface.py:

from PyQt5 import QtGui

from PyQt5 import QtWidgets

from PyQt5 import QtCore

from PyQt5.QtCore import QSize, Qt, pyqtSlot

from PyQt5.QtGui import QFont, QTextLine

from PyQt5.QtWidgets import QApplication, QDialog, QLabel, QLineEdit, QMainWindow, QPushButton, QTableWidget, QTableWidgetItem, \

    QWidget

import sys

sys.path.insert(0, './Program')

from database import MyDb

import sqlite3

class MainWindow(QMainWindow):

    def \_\_init\_\_(self):

        """

            Инициализаци окна программы.

        """

        super().\_\_init\_\_()

        self.db = MyDb(sqlite3.connect(sys.path[1]+'/db.sqlite'))

        self.setWindowTitle("IMag")

        self.setFixedSize(QSize(1000, 600))

        self.login\_area = QWidget(self)

        self.setCentralWidget(self.login\_area)

        self.login\_text = QLabel("Войти", self.login\_area)

        self.login\_text.setGeometry(440, 224, 120, 16)

        self.login\_text.setAlignment(Qt.AlignCenter)

        self.login\_linedit = QLineEdit("", self.login\_area)

        self.login\_linedit.setGeometry(440, 240, 120, 30)

        self.password\_text = QLabel("Password", self.login\_area)

        self.password\_text.setGeometry(440, 274, 120, 16)

        self.password\_text.setAlignment(Qt.AlignCenter)

        self.password\_lineedit = QLineEdit("", self.login\_area)

        self.password\_lineedit.setGeometry(440, 290, 120, 30)

        self.password\_lineedit.setEchoMode(QLineEdit.Password)

        self.authorize\_button = QPushButton("Log in", self.login\_area)

        self.authorize\_button.setGeometry(460, 330, 80, 40)

        self.authorize\_button.clicked.connect(self.login)

        self.show()

    def login(self):

        """

            Авторизация при помощи запроса к базе данных.

        """

        response = self.db.check\_user\_in\_authorization(self.login\_linedit.text(), self.password\_lineedit.text())

        print(self.login\_linedit.text(), self.password\_lineedit.text(), response)

        self.db.debug\_print('Authorization')

        if response == 1:

            self.initialize\_main\_window()

            print("Password is correct. Opening main view.")

        else:

            self.password\_lineedit.setText('')

            print("Password is not correct. Login:", self.login\_linedit.text(),

                  ", Password:" + self.password\_lineedit.text())

    def initialize\_main\_window(self):

        """

            Загрузка элементов главной таблицы.

        """

        self.login\_area.setParent(None)

        print('Initialized.')

        self.main\_area = QWidget(self)

        self.setCentralWidget(self.main\_area)

        self.main\_table = QTableWidget(self.main\_area)

        self.main\_table.setGeometry(20, 80, 960, 500)

        self.main\_table.setSizeAdjustPolicy(QtWidgets.QAbstractScrollArea.AdjustToContents)

        self.main\_table.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn);

        self.history\_button = QPushButton('Все покупки', self.main\_area)

        self.history\_button.setGeometry(20, 20, 100, 25)

        self.history\_button.clicked.connect(self.show\_history)

        self.bought\_products\_button = QPushButton('Найти покупки', self.main\_area)

        self.bought\_products\_button.setGeometry(130, 20, 230, 25)

        self.bought\_products\_button.clicked.connect(self.show\_bought\_products)

        self.bought\_products\_name\_lineedit = QLineEdit(self.main\_area)

        self.bought\_products\_name\_lineedit.setPlaceholderText('Имя покупателя')

        self.bought\_products\_name\_lineedit.setGeometry(131, 47, 113, 25)

        self.bought\_products\_name\_lineedit.setAlignment(Qt.AlignCenter)

        self.bought\_products\_surname\_lineedit = QLineEdit(self.main\_area)

        self.bought\_products\_surname\_lineedit.setPlaceholderText('Фамилия покупателя')

        self.bought\_products\_surname\_lineedit.setGeometry(246, 47, 113, 25)

        self.bought\_products\_surname\_lineedit.setAlignment(Qt.AlignCenter)

        self.product\_buyers\_button = QPushButton('Найти покупателей', self.main\_area)

        self.product\_buyers\_button.setGeometry(369, 20, 230, 25)

        self.product\_buyers\_button.clicked.connect(self.show\_product\_buyers)

        self.product\_buyers\_article\_lineedit = QLineEdit(self.main\_area)

        self.product\_buyers\_article\_lineedit.setPlaceholderText('Артикул')

        self.product\_buyers\_article\_lineedit.setGeometry(370, 47, 229, 25)

        self.product\_buyers\_article\_lineedit.setAlignment(Qt.AlignCenter)

        self.show\_buyers\_button = QPushButton('Показать покупателей', self.main\_area)

        self.show\_buyers\_button.setGeometry(609, 20, 230, 25)

        self.show\_buyers\_button.clicked.connect(self.show\_buyers)

        self.add\_buyers\_button = QPushButton('Добавить покупателя', self.main\_area)

        self.add\_buyers\_button.setGeometry(609, 47, 230, 25)

        self.add\_buyers\_button.clicked.connect(self.show\_addbuyer\_dialog)

    def show\_history(self):

        """

            Загрзка записей покупок в окно главной таблицы.

        """

        history = self.db.select\_all()

        self.show\_table\_data(history,

                             ['Название товара', 'Цена', 'Артикул', 'Имя покупателя', 'Фамилия покупателя', 'Скидка', 'Телефон','Клубный статус',

                              'Название магазина', 'Адрес магазина', 'Дата покупки'])

    def show\_bought\_products(self):

        """

            Загрзка записей о покупках в окно главной таблицы.

        """

        bought\_products = self.db.get\_by\_name(self.bought\_products\_name\_lineedit.text(),

                                              self.bought\_products\_surname\_lineedit.text())

        self.show\_table\_data(bought\_products,

                             ['Название товара', 'Цена', 'Арткул', 'Название магазина', 'Адрес магазина',

                              'Дата покупки'])

    def show\_product\_buyers(self):

        """

            Загрзка записей о покупателях для конкретного товара в окно главной таблицы.

        """

        bought\_products = self.db.get\_by\_article(self.product\_buyers\_article\_lineedit.text())

        self.show\_table\_data(bought\_products,

                             ['Название товара', 'Цена', 'Фамилия покупателя', 'Имя покупателя', 'Скидка', 'Телефон','Клубный статус',

                              'Название магазина', 'Адрес магазина', 'Дата покупки'])

    def show\_buyers(self):

        """

            Загрзка записей о всех покупателях и их клубных картах в окно главной таблицы.

        """

        bought\_products = self.db.get\_buyers()

        self.show\_table\_data(bought\_products,

                             ['Имя покупателя', 'Фамилия покупателя', 'Скидка', 'Телефон','Клубный статус'])

    def show\_table\_data(self, data, lables=['']):

        """

            Преобразование ответа базы данных в строки таблицы и их вывод на экран.

        """

        self.clear\_main\_table()

        if (len(data)) == 0:

            self.display\_error()

            print('No data found to fill the table.')

            return

        self.main\_table.verticalHeader().setVisible(True)

        self.main\_table.horizontalHeader().setVisible(True)

        self.main\_table.setColumnCount(len(data[0]))

        for i in range(len(data)):

            self.main\_table.insertRow(i)

            entry = data[i]

            for j in range(len(entry)):

                item = QTableWidgetItem(str(entry[j]))

                item.setFlags(QtCore.Qt.ItemIsEnabled)

                self.main\_table.setItem(i, j, item)

        self.main\_table.setHorizontalHeaderLabels(lables)

        self.main\_table.resizeColumnsToContents()

    def display\_error(self):

        """

            Вывод сообщения об отсутсвии записей в ответном сообщении.

        """

        self.main\_table.setColumnCount(1)

        self.main\_table.insertRow(0)

        self.main\_table.setItem(0, 0, QTableWidgetItem('Информация не найдена'))

        self.main\_table.resizeColumnsToContents()

        self.main\_table.verticalHeader().setVisible(False)

        self.main\_table.horizontalHeader().setVisible(False)

    def clear\_main\_table(self):

        """

            Очистка главной таблицы.

        """

        self.main\_table.clearContents()

        while self.main\_table.rowCount() > 0:

            self.main\_table.removeRow(0)

        while self.main\_table.columnCount() > 0:

            self.main\_table.removeColumn(0)

    def show\_addbuyer\_dialog(self):

        """

            Вывод диалогового окна добавления покупателя.

        """

        self.dialog = QDialog()

        self.dialog.setAttribute(QtCore.Qt.WA\_DeleteOnClose)

        self.dialog.setWindowTitle("Добавление покупателя")

        self.dialog.setFixedSize(QSize(400, 300))

        self.dialog.paramEditBoxes = []

        self.dialog.paramNameLabels = []

        paramNames = ['Имя покупателя', 'Фамилия покупателя', 'Скидка', 'Телефон', 'Клубный статус']

        for i in range(0, len(paramNames)):

            self.add\_dialog\_param(self.dialog, i, paramNames[i])

        self.dialog.add\_buyer\_button = QPushButton('Добавить', self.dialog)

        self.dialog.add\_buyer\_button.setGeometry(150, 265, 100, 25)

        self.dialog.add\_buyer\_button.clicked.connect(self.add\_buyer)

        self.dialog.show()

        self.dialog.exec()

    def add\_buyer(self):

        """

            Добавление покупателя

        """

        params = self.dialog.paramEditBoxes

        self.db.insert\_into\_buyers(params[0].text(), params[1].text(), params[2].text(), params[3].text(), params[4].text())

    def add\_dialog\_param(self, dialog, i, paramName):

        """

            Загрузка полей в диалоговое окно добавления покупателя.

        """

        text = QLabel(paramName, dialog)

        text.setGeometry(10, 10+i\*40, 165, 20)

        text.setAlignment(Qt.AlignCenter)

        lineedit = QLineEdit("", dialog)

        lineedit.setGeometry(175, 10+i\*40, 165, 20)

        dialog.paramEditBoxes.append(lineedit)

        dialog.paramNameLabels.append(text)

        self.dialog.close()

def init():

    """

        Запуск окна программы.

    """

    app = QApplication(sys.argv)

    window = MainWindow()

    window.login\_linedit.setText('admin')

    window.password\_lineedit.setText('1111')

    app.exec()

if \_\_name\_\_ == "\_\_main\_\_":

    init()

Модуль database.py:

import sqlite3

class MyDb():

    def \_\_init\_\_(self, bd):

        self.bd = bd

        self.curr = bd.cursor()

        self.curr.execute("""

                create table if not exists Authorization(

                    authorization\_id integer primary key autoincrement,

                    login char(16),

                    password char(16)

                )""")

        self.curr.execute("""

                create table if not exists Products(

                    product\_id integer primary key autoincrement,

                    name char(16),

                    price integer,

                    article char(20)

                )""")

        self.curr.execute("""

                create table if not exists Buyers(

                    buyer\_id integer primary key autoincrement,

                    name char(16),

                    surname char(16),

                    discount double,

                    phone char(16)

                )""")

        self.curr.execute("""

                create table if not exists Store(

                    store\_id integer primary key autoincrement,

                    name char(16),

                    address char(64)

                )""")

        self.curr.execute("""

                create table if not exists History(

                    history\_id integer primary key autoincrement,

                    product\_id integer,

                    buyer\_id integer,

                    store\_id integer,

                    buy\_date date,

                    foreign key (product\_id) references products (product\_id),

                    foreign key (buyer\_id) references buyers (buyer\_id),

                    foreign key (store\_id) references store (store\_id)

                    )""")

        self.bd.commit()

    def select\_all(self):

        """

        Возвращает список кортежей, в каждом кортеже указаны все данные.

        """

        self.curr.execute("""

        SELECT Products.name,Products.price,article,Buyers.name,Buyers.surname,

        Buyers.discount,Buyers.phone,Buyers.status,Store.name,Store.address,buy\_date FROM History

        join Products on History.product\_id=Products.product\_id

        join Buyers on History.buyer\_id=Buyers.buyer\_id

        join Store on History.store\_id=Store.store\_id

        """)

        return self.curr.fetchall()

    def get\_buyers(self):

        """

        Таблица покупателей.

        Возвращает список из кортежей, где каждый кортеж - Имя, Фамилимя,скидка, телефон, клубный статус

        """

        self.curr.execute("""

        select name, surname,discount,phone,status from Buyers

        """)

        return self.curr.fetchall()

    def get\_by\_name(self, name, surname):

        """

        Принимает две строки - имя и фамилия, по которым осуществляется фильтрация.

        Возвращает список из кортежей, где каждый кортеж -  Имя товара, цена, арткул, имя магазина, адрес магазина,

        дата покупки.

        """

        self.curr.execute(f"""

        SELECT Products.name,Products.price,article,Store.name,Store.address,buy\_date FROM History

        join Products on History.product\_id=Products.product\_id

        join Buyers on History.buyer\_id=Buyers.buyer\_id

        join Store on History.store\_id=Store.store\_id

        where Buyers.name='{name}' and Buyers.surname='{surname}'""")

        return self.curr.fetchall()

    def get\_by\_article(self, article):

        """

        Принимает строку - артикул, по которому осуществляется фильтрация

        Возвращает список из кортежей, где каждый кортеж - имя товара,цену товара,имя покупателя, фамилиюю покупателя,

        его скидка, имя магазина, адрес магазина,

        дату покупки.

        """

        self.curr.execute(f"""SELECT Products.name,Products.price,Buyers.name,Buyers.surname,Buyers.discount,Store.name,

        Store.address,buy\_date FROM History

        join Products on History.product\_id=Products.product\_id

        join Buyers on History.buyer\_id=Buyers.buyer\_id

        join Store on History.store\_id=Store.store\_id

        where Products.article='{article}'""")

        return self.curr.fetchall()

    def check\_user\_in\_authorization(self, login, password):

        """

        Принимает строки - логин и пароль

        Возвращает 1, если пользователь есть в таблице авторизации

        Возвращает 0, если пользователя нет в таблице авторизации

        """

        self.curr.execute(f"""select count(login) from Authorization

                                where login = '{login}' and password='{password}'""")

        user = self.curr.fetchall()[0][0]

        if user == 1:

            return 1

        elif user == 0:

            return 0

    def insert\_into\_buyers(self, name, surname, discount, phone, status):

        """

        Заносит запись в таблицу с информаицей о покупателях

        :param name: str

        :param surname: str

        :param discount: float

        :param phone: str

        :param status: str

        :return: None

        """

        self.curr.execute(

            f"""INSERT INTO Buyers(name,surname,discount,phone,status) VALUES('{name}','{surname}',{discount},'{phone}','{status}')""")

        self.bd.commit()

    def close\_con(self):

        """Закрывает соединение с базой"""

        self.bd.close()

    def debug\_print(self, tablename):

        request = f"""SELECT \* FROM {tablename}"""

        print('Request:', request)

        self.curr.execute(request)

        print(f"{tablename} contents:", self.curr.fetchall())

if \_\_name\_\_ == '\_\_main\_\_':

    db = MyDb(sqlite3.connect('db.sqlite'))

    print(db.get\_buyers())