Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский экономический университет имени Г.В. Плеханова»
МОСКОВСКИЙ ПРИБОРОСТРОИТЕЛЬНЫЙ ТЕХНИКУМ.

Специальность: 09.02.07 Информационные системы и программирование

Квалификация: Программист

ПРАКТИЧЕСКАЯ РАБОТА

ПО «ОАИП Python»

Разработка информационной системы

Выполнил(а) студент(ка)
Группы П50-6-22
Мурлаева Екатерина Олеговна
«____» _____ 2023 года

Проверил преподаватель
_____Т.Д. Артамонова
«____» _____2023 года

Москва 202

Цель работы: Разработать информационную систему по своей предметной области, используя инкапсуляция, наследование и полиморфизм

Ход работы:

1. Создание кода

Код представляет собой простую систему управления базой данных для магазина телефонов.

В нем реализованы четыре класса: Database, User, Employee, и Tovar

Класс Database:

Данный класс обеспечивает взаимодействие с базой данных, создание таблиц, добавление данных и другие операции.

Основные функции класса Database:

- Создание таблиц для пользователей (users), сотрудников (employees), товаров (tovars), и заказов (orders)
- Методы для добавления пользователей, сотрудников, товаров и заказов
- Методы для удаления таблицы товаров и получения всех товаров из базы данных
- Методы для создания начальных записей о товарах и получения списка всех товаров

```
class Database:

    def __init__(self):
        self.conn = sqlite3.connect('phone_store_db.db')
        self.cursor = self.conn.cursor()
        self.create_tables()

    def create_tables(self):
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS users (
                id INTEGER PRIMARY KEY,
                username TEXT UNIQUE,
                password TEXT,
                role TEXT,
                full_name TEXT
            )
        ''')
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS employees (
                id INTEGER PRIMARY KEY,
                username TEXT UNIQUE,
                password TEXT,
                full_name TEXT
            )
        ''' )
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS tovars (
                id INTEGER PRIMARY KEY,
                name TEXT UNIQUE,
                price INTEGER
            )
        ''')
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS orders (
                id INTEGER PRIMARY KEY,
                user_id INTEGER,
                tovar_id INTEGER,
                FOREIGN KEY (user_id) REFERENCES users (id),
                FOREIGN KEY (tovar_id) REFERENCES tovars (id)
            )
        ''')
        self.conn.commit()
```

Рисунок 1 - Класс Database

```
def add_user(self, username, password, role, full_name):
    hashed_password = hashlib.sha256(password.encode()).hexdigest()
    self.cursor.execute('''
        INSERT INTO users (username, password, role, full_name)
        VALUES (?, ?, ?, ?)
    ''', (username, hashed_password, role, full_name))
    self.conn.commit()

def get_user_by_username(self, username):
    self.cursor.execute('SELECT * FROM users WHERE username = ?', (username,))
    return self.cursor.fetchone()

def add_employee(self, username, password, full_name):
    hashed_password = hashlib.sha256(password.encode()).hexdigest()
    self.cursor.execute('''
        INSERT INTO employees (username, password, full_name)
        VALUES (?, ?, ?)
    ''', (username, hashed_password, full_name))
    self.conn.commit()

def add_tovar(self, name, price):
    self.cursor.execute('''
        INSERT INTO tovars (name, price)
        VALUES (?, ?)
    ''', (name, price))
    self.conn.commit()

def drop_tovars_table(self):
    self.cursor.execute('DROP TABLE IF EXISTS tovars')
    self.conn.commit()

def get_all_tovars(self):
    self.cursor.execute('SELECT * FROM tovars')
    rows = self.cursor.fetchall()
    tovars = []
    for row in rows:
        tovar = Tovar(row[1], row[2])
        tovars.append(tovar)
    return tovars

def add_order(self, user_id, tovar_id):
    self.cursor.execute('''
        INSERT INTO orders (user_id, tovar_id)
        VALUES (?, ?)
    ''', (user_id, tovar_id))
    self.conn.commit()
```

Рисунок 2 – Класс Database

Классы User и Employee:

Данные классы предоставляют атрибуты для хранения информации (ID, логин, пароль, полное имя) о пользователе и сотруднике.

```
class User:
    def __init__(self,user_id, username, password, role, full_name):
        self.id = user_id
        self.username = username
        self.password = password
        self.role = role
        self.full_name = full_name
```

Рисунок 3 – Класс User

```
class Employee(User):
    def __init__(self, user_id, username, password, full_name):
        super().__init__(user_id, username, password, 'Employee', full_name)
```

Рисунок 4 – Класс Employee

Класс Tovar:

Данный класс схож с двумя предыдущими, но предоставляет атрибуты для хранения информации (название, цена) о товаре в магазине.

```
class Tovar:
    def __init__(self, name, price):
        self.name = name
        self.price = price
```

Рисунок 5 - Класс Tovar

2. Функционал кода:

- Регистрация и Вход

Пользователи могут зарегистрироваться, указав логин, пароль, роль (клиент, сотрудник, админ) и полное имя.

```python
def register_user(db):
    username = input("Введите логин: ")
    password = input("Введите пароль: ")
    role = input("Выберите роль:\n 1 - Клиент\n 2 - Сотрудник\n 3 - Админ\n "
    full_name = input("Введите ваше полное имя: ")

    if role == "1":
        role = "Клиент"
    elif role == "2":
        role = "Сотрудник"
    elif role == "3":
        role = "Админ"
    else:
        print("Неверная роль.")
        return
    db.add_user(username, password, role, full_name)
    print("Регистрация успешна!")
```

Рисунок 6 – Регистрация

- Пользователи могут войти в систему, указав свой логин и пароль.

```python
def login_user(db):
    username = input("Введите логин: ")
    password = input("Введите пароль: ")

    user_data = db.get_user_by_username(username)

    if user_data and hashlib.sha256(password.encode()).hexdigest() == user_data[2]:
        if user_data[3] == "Клиент":
            return User(user_data[1], user_data[2], user_data[3], user_data[4])
        elif user_data[3] == "Сотрудник":
            return Employee(user_data[1], user_data[2], user_data[3], user_data[4])
    else:
        print("Неверный логин или пароль.")
    return None
```

Рисунок 7 – Вход в систему

- Меню для клиента, сотрудника и админа

У каждого пользователя, в зависимости от того, какую роль он выбрал, будет появляться свое меню с разными выборами действий

- Клиент может просматривать товары, добавлять заказы, просматривать свои заказы, изменять свои данные и выйти в меню регистрации, входа.

```python
def client_menu(user, db):
        while True:
            print("1. Просмотр товаров")
            print("2. Добавить заказ")
            print("3. Просмотр заказов")
            print("4. Изменить данные")
            print("5. Выйти")

            choice = input("Выберите действие: ")

            if choice == "1":
                show_tovars(db)
            elif choice == "2":
                add_order(user, db)
            elif choice == "3":
                show_orders(user, db)
            elif choice == "4":
                update_user_data(user, db)
            elif choice == "5":
                break
            else:
                print("Неверный ввод. Попробуйте снова.")
```

Рисунок 8 – Меню клиента

- Сотрудник может просматривать товары, добавлять товары, удалять товары, изменять свои данные и выйти в меню регистрации, входа.

```python
def employee_menu(user, db):
        while True:
            print("1. Просмотр товаров")
            print("2. Добавить товар")
            print("3. Удалить товар")
            print("4. Изменить данные")
            print("5. Выйти")

            choice = input("Выберите действие: ")

            if choice == "1":
                show_tovars(db)
            elif choice == "2":
                add_tovar(db)
            elif choice == "3":
                delete_tovar(db)
            elif choice == "4":
                update_user_data(user, db)
            elif choice == "5":
                break
            else:
                print("Неверный ввод. Попробуйте снова.")
```

Рисунок 9 – меню сотрудника

- Администратор может просматривать сотрудников, добавлять сотрудников, изменять свои данные и выйти в меню регистрации, входа.

```python
def admin_menu(user, db):
        while True:
            print("1. Просмотр всех сотрудников")
            print("2. Добавить сотрудника")
            print("3. Изменить данные")
            print("4. Выйти")

            choice = input("Выберите действие: ")

            if choice == "1":
                show_employees(db)
            elif choice == "2":
                add_employee(db)
            elif choice == "3":
                update_user_data(user, db)
            elif choice == "4":
                break
            else:
                print("Неверный ввод. Попробуйте снова.")
```

Рисунок 10 – меню админа

3. Дополнительные функции

В зависимости от роли у каждого свое меню, где разный выбор действий, но у всех них есть одинаковая функция – обновление данных

```python
def update_user_data(user, db):
        new_username = input("Введите новый логин: ")
        new_password = input("Введите новый пароль: ")

        table_name = "users"

        query = f'''
            UPDATE {table_name} SET password = ?, full_name = ? WHERE id = ?
        '''
        db.cursor.execute(query, (hashlib.sha256(new_password.encode()).hexdigest(), new_username, user[0]
        db.conn.commit()
        print("Данные обновлены успешно!")
```

Рисунок 11 – Обновление данных

- Клиенты

```
def show_tovars(db):
        tovars = db.cursor.execute('SELECT * FROM tovars').fetchall()
        print("Список товаров:")
        for tovar in tovars:
            print(f"{tovar[0]}. {tovar[1]} - {tovar[2]} руб.")
```

Рисунок 12 – Просмотр товаров

```
def add_order(user, db):
        show_tovars(db)
        tovar_id = int(input("Введите ID товара для заказа: "))

        db.add_order(user[2], tovar_id)
        print("Заказ оформлен успешно!")
```

Рисунок 13 – Создание заказа, добавление туда товара

```
def show_orders(user, db):
    orders = db.cursor.execute('SELECT * FROM orders WHERE user_id = ?', (user[2],)).
    print("Ваши заказы:")
    for order in orders:
        tovar = db.cursor.execute('SELECT * FROM tovars WHERE id = ?', (order[2],)).f
        print(f"{order[0]}. {tovar[1]} - {tovar[2]} руб.")
```

Рисунок 14 – Просмотр товаров в заказе

- Сотрудники

```
def show_tovars(db):
        tovars = db.cursor.execute('SELECT * FROM tovars').fetchall()
        print("Список товаров:")
        for tovar in tovars:
            print(f"{tovar[0]}. {tovar[1]} - {tovar[2]} руб.")
```

Рисунок 15 – Просмотр товаров

```
def add_tovar(db):
    name = input("Введите название товара: ")
    price = int(input("Введите цену товара: "))
    if not isinstance(price, (int)) or price < 2000:
            print("Ошибка: Цена должна быть быть больше.")
            return

    db.add_tovar(name, price)
    print("Товар добавлен успешно!")
```

Рисунок 16 – Добавление товаров в список

В данной функции(add_tovar) также идет проверка на то, чтоб цена добавленного товара было не менее 2000руб.

```
def delete_tovar(db):
        show_tovars(db)
        tovar_id = int(input("Введите ID товара для удаления: "))

        db.cursor.execute('DELETE FROM tovars WHERE id = ?', (tovar_id,))
        db.conn.commit()
        print("Товар удален успешно!")
```

Рисунок 17 - Удаление товара из списка

- Админ

```
def show_employees(db):
        employees = db.cursor.execute('SELECT * FROM users WHERE role = "Сотрудник"').
        print("Список сотрудников:")
        for employee in employees:
            print(f"{employee[0]}. {employee[4]} ({employee[1]}) - {employee[2]}")
```

Рисунок 18 – Просмотр списка сотрудников

```
def add_employee(db):
        username = input("Введите логин сотрудника: ")
        password = input("Введите пароль сотрудника: ")
        full_name = input("Введите имя сотрудника: ")

        db.add_employee(username, password, full_name)
        print("Сотрудник добавлен успешно!")
```

Рисунок 19 – Добавление нового сотрудника

4. Функция main

В данном функции прописывается интерфейс в консоли, там вызывается функции для регистрации и авторизации, после в зависимости от роли у вас выполняются те или иные функции.

```python
def main():
    db = Database()

    while True:
        print("1. Регистрация")
        print("2. Вход")
        print("3. Выход")
        choice = input("Выберите действие: ")

        if choice == "1":
            register_user(db)
        elif choice == "2":
            username = input("Введите логин: ")
            password = input("Введите пароль: ")
            user = db.get_user_by_username(username)

            if user and user[2] == hashlib.sha256(password.encode()).hexdigest():
                print(f"Добро пожаловать, {user[4]}!")

                if user[3] == "Клиент":
                    client_menu(user,db)
                elif user[3] == "Сотрудник":
                    employee_menu(user,db)
                elif user[3] == "Админ":
                    admin_menu(user,db)
                else:
                    print("Неверная роль.")
            else:
                print("Неверный логин или пароль.")
        elif choice == "3":
            break
        else:
            print("Неверный выбор.")
```

Рисунок 20 – Функция main

5. Инкапсуляция, наследование и полиморфизм

- Инкапсуляция
    - Класс Database

В данном примере conn и cursor являются приватными атрибутами класса, так как они не являются доступными извне класса. Они предоставляют интерфейс для взаимодействия с базой данных, но детали реализации скрыты.

- Метод add_user в классе Database

Здесь hashed_password - это локальная переменная, инкапсулированная внутри метода. Она не доступна извне класса и используется только внутри метода add_user.

- Атрибуты классов User, Employee и Tovar

В каждом из этих классов атрибуты, такие как id, username, password, role, и full_name, инкапсулированы внутри экземпляров класса и могут быть доступны только через методы и свойства классов.

- Наследование
  - Класс Employee наследует от класса User

В этом примере класс Employee является подклассом класса User. Он наследует все атрибуты и методы класса User, добавляя при этом свой собственный функционал. Использование super().__init__ в конструкторе Employee также обеспечивает вызов конструктора базового класса User.

Таким образом, объекты класса Employee могут использовать функционал класса User и в то же время иметь свои уникальные атрибуты и методы.

- Полиморфизм
  - Методы меню для клиента, сотрудника и админа

Они все обладают одинаковым интерфейсом для работы с разными ролями (клиент, сотрудник, админ). Каждый из этих методов реализует свою уникальную логику в соответствии с ролью пользователя.

- Методы show_tovars, add_tovar, delete_tovar

В этих методах используются общие интерфейсы для работы с товарами, независимо от роли пользователя.

- Классы User и Employee

  Класс Employee наследует от класса User и переопределяет метод __init__, т.е объект класса Employee может использоваться в тех местах, где ожидается объект класса User.

Вывод: в ходе работы была разработать информационная систему по своей предметной области, используя инкапсуляция, наследование и полиморфизм