Kursinio darbo ataskaita

Atliko: Mindaugas Moisiejus EDIF-24/2

Aprašo pradžia

Mano kursinio darbo programos tikslas buvo padėti gydytojams, medikams su ligoninės tvarka bei pagerinti jų efektyvumą. Dabartinis funkcionalumas apima:

1. Ligoninės personalo tvarkymas;
2. Ligoninės kambarių tvarkymas;
3. Funkcija suteikti žinias apie ligonines, kurios gali priimti ir laikyti pacientus iš pirmosios pagalbos.

Naudotis šia programėle galima įjungus *main.exe* failą dist direktorijoje. Joje yra mygtukai pro, kuriuos galima atlikti anksčiau minėtas funkcijas (viskas yra anglų kalba).

Kursinio darbo reikalavimų įvykdymas

Visi darbo failai yra įkelti šioje github svetainėje.

Reikėjo įgyvendinti visus objektinio programavimo principus:

1. *Polimorphism* – įgyvendintas naudojant funkcija grižtant pro UI ekranus. Kiekvienas go_back metodas buvo *overrided* naudojant kitą reikšmę.
2. *Abstraction* – demonstruojama *MedicalStaff* klasėje ir iš jos kilusiomis klasėmis *Nurse* ir *Doctor.*
3. *Inheritance* – demonstruojama *MedicalStaff*, *Nurse* ir *Doctor* klasių. Kiekvienas UI ekranas paveldi iš BasePage *go_back* metodą.
4. *Encapsulation – Hospital* klasės sąrašai, pavadinimas bei tų pačių *MedicalStaff*, *Nurse* ir *Doctor* klasių reikšmės yra *protected*.

Buvo naudojamas *Factory Method Design Pattern*, kuris sukuria objektus *Doctor*, *Nurse*, *Room*.

Kompozicija *Hospital* klasėje. *Doctor*, *Nurse* ir *Room* objektai egzistuoja tik tada, kai ir *Hospital* klasės objektas egzistuoja.

Duomenys traukiami ir saugomi į .txt failus.

Kodas buvo testuotas naudojant *unittest framework*. Testavimo kodas yra *test* direktorijoje.

Kodas buvo parašytas atsižvelgiant į *PEP8* stilių.

Buvo ištestuota naudojant *unittest* bei savo sukurtais scenarijais.

(Kodas demonstruojantis kiekvieną reikalavimą galę)

Rezultatai

- Programa geba parodyti kurios ligoninės turi laisvų vietų.
- Programa geba pridėti prie ir išimti iš ligoninės sąrašo personalą bei kambarius.
- Programa geba išsaugoti į .txt failus duomenis, kurie vėliau yra naudojami.
- Programa geba ištrinti ligonines iš sąrašo.
- Programuojant išmokau naudotis *tkinter* gui įgyvendinimo biblioteka, išsaugojimo *os* biblioteka *Python* kalboje.

Išvados

Sugebėjau sukurti pagrindą programos, kuri gali palengvinti daktarų darbą.

Dirbant prie šio kursinio darbo pagilinau savo žinias naudojant *Python* kalbą bei sužinojau kaip naudotis (šiek tiek) bibliotekomis *tkinter* ir *os*.

Šiai programai galima pridėti daugiau funkcijų kaip užimti daktarus kai jų operacijų laikas sutampa, vaistų organizavimo funkciją, *GPS* integravimą su ligoninėmis, kurios gali priimti pacientų.

Github svetainė:

https://github.com/RedFlake-stud/UniversityProject_me

Polimorphism:

main.py base_page.py

ui_elements > base_page.py > BasePage > go_back

```python
import tkinter as tk


class BasePage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller

    def go_back(self, target_page):
        self.controller.show_frame(target_page)
```

```python
    def go_back(self):
        from ui_elements.config_page import ConfigPage
        super().go_back(ConfigPage)
```

Abstraction, Inheritance:

models > staff.py > Doctor

```python
class MedicalStaff:
    def __init__(self, name, age, gender):
        self._name = name
        self._age = age
        self._gender = gender


class Doctor(MedicalStaff):
    def __init__(self, name, age, gender, specialization):
        super().__init__(name, age, gender)
        self._specialization = specialization
        self._status = "Available"

    def update_status(self, new_status):
        self._status = new_status

    def __str__(self):
        return f"Doctor: {self._name} ({self._specialization})"


class Nurse(MedicalStaff):
    def __init__(self, name, age, gender):
        super().__init__(name, age, gender)
        self._status = "Available"

    def update_status(self, new_status):
        self._status = new_status

    def __str__(self):
        return f"Nurse: {self._name}"
```

Encapsulation:

```python
class Hospital:
    def __init__(self, name, filename=None):
        self._name = name
        self._doctors = []
        self._nurses = []
        self._rooms = []
        self._filename = filename
```

Factory Method:

```python
from models.staff import Doctor, Nurse
from models.room import Room


class Factory:
    @staticmethod
    def create_doctor(name, age, gender, specialization, status="Available"):
        doctor = Doctor(name, age, gender, specialization)
        doctor.update_status(status)
        return doctor

    @staticmethod
    def create_nurse(name, age, gender, status="Available"):
        nurse = Nurse(name, age, gender)
        nurse.update_status(status)
        return nurse

    @staticmethod
    def create_room(room_number, capacity, patients):
        return Room(room_number, capacity, patients)
```

Composition:

```python
class Hospital:
    def __init__(self, name, filename=None):
        self._name = name
        self._doctors = []
        self._nurses = []
        self._rooms = []
        self._filename = filename

    def add_doctor(self, doctor):
        for existing_doctor in self._doctors:
            if existing_doctor._name == doctor._name:
                print("Doctor already exists. Updating data instead...")
                self.remove_doctor(existing_doctor)
        self._doctors.append(doctor)
        self.save_to_file()

    def add_nurse(self, nurse):
        for existing_nurse in self._nurses:
            if existing_nurse._name == nurse._name:
                print("Nurse already exists. Updating data instead...")
                self.remove_nurse(existing_nurse)
        self._nurses.append(nurse)
        self.save_to_file()

    def add_room(self, room):
        for existing_room in self._rooms:
            if existing_room._room_number == room._room_number:
                print("Room already exists. Updating data instead...")
                self.remove_room(existing_room)
        self._rooms.append(room)
        self.save_to_file()
```

Reading/writing from/to file:

```python
class HospitalManager:
    def __init__(self, data_dir="hospital_data"):
        self._hospitals = {}
        self._data_dir = data_dir
        os.makedirs(data_dir, exist_ok=True)

    def add_hospital(self, hospital):
        self._hospitals[hospital._name] = hospital
        self.save_hospital(hospital)

    def get_hospital(self, name):
        return self._hospitals.get(name, None)

    def list_hospitals(self):
        return list(self._hospitals.keys())

    def save_hospital(self, hospital):
        filename = os.path.join(self._data_dir, f"{hospital._name}.txt")
        hospital._filename = filename
        hospital.save_to_file()

    def load_all_hospitals(self):
        for file in os.listdir(self._data_dir):
            if file.endswith(".txt"):
                name = file[:-4]
                hospital = Hospital(name)
                hospital._filename = os.path.join(self._data_dir, file)
                hospital.load_from_file()
                self._hospitals[name] = hospital

    def delete_hospital(self, hospital):
        if hospital._name in self._hospitals:
            del self._hospitals[hospital._name]
            os.remove(hospital._filename)
```

```python
class AvailableHospitals(BasePage):
```

```python
    def update_available_hospitals(self):
        self.hospitals_list.delete(0, tk.END)
        for name, hospital in self.manager._hospitals.items():
            print(f"Checking availability for {name}: {hospital.is_available()}")
            if hospital.is_available():
                self.hospitals_list.insert(tk.END, name)
```

Unittest (1 pavyzdys):

```python
import unittest
from hospital_manager import Hospital
from models.staff import Doctor, Nurse
from models.room import Room


class TestHospital(unittest.TestCase):
    def setUp(self):
        self.hospital = Hospital("Test Hospital", filename="test_hospital

    def test_add_doctor(self):
        doctor = Doctor("John Doe", 40, "Male", "Cardiology")
        self.hospital.add_doctor(doctor)
        self.assertIn(doctor, self.hospital._doctors)

    def test_add_nurse(self):
        nurse = Nurse("Jane Smith", 30, "Female")
        self.hospital.add_nurse(nurse)
        self.assertIn(nurse, self.hospital._nurses)

    def test_add_room(self):
        room = Room("101", 2, 1)
        self.hospital.add_room(room)
        self.assertIn(room, self.hospital._rooms)

    def test_is_available(self):
        doctor = Doctor("John Doe", 40, "Male", "Cardiology")
        nurse1 = Nurse("Jane Smith", 30, "Female")
        nurse2 = Nurse("Alice Brown", 35, "Female")
        room = Room("101", 2, 1)

        self.hospital.add_doctor(doctor)
        self.hospital.add_nurse(nurse1)
        self.hospital.add_nurse(nurse2)
        self.hospital.add_room(room)

        self.assertTrue(self.hospital.is_available())


if __name__ == "__main__":
    unittest.main()
```