Технологии конструирования программного обеспечения
Отчет по лабораторной работе № 01
Группа: 221-329 **Студент:** Минчаков Аркадий Сергеевич

Задание на лабораторную работу:

Для заданного варианта задания разработать UML-диаграмму классов и диаграмму последовательности. Разработать программу решения задания в виде консольного приложения (C++, C#) с использованием принципа единственной обязанности (SRP) и принципа открытости/закрытости (OCP). Допускается вводить дополнительные понятия предметной области. В программе предусмотрите тестирование функциональности созданных объектов классов.

Номер в списке - 15, вариант 3:

Система Больница. Пациенту назначается лечащий Врач. Врач может сделать назначение Пациенту (процедуры, лекарства, операции). Медсестра или другой Врач выполняют назначение. Пациент может быть выписан из Больницы по окончании лечения, при нарушении режима или при иных обстоятельствах.

UML-диаграмма классов:

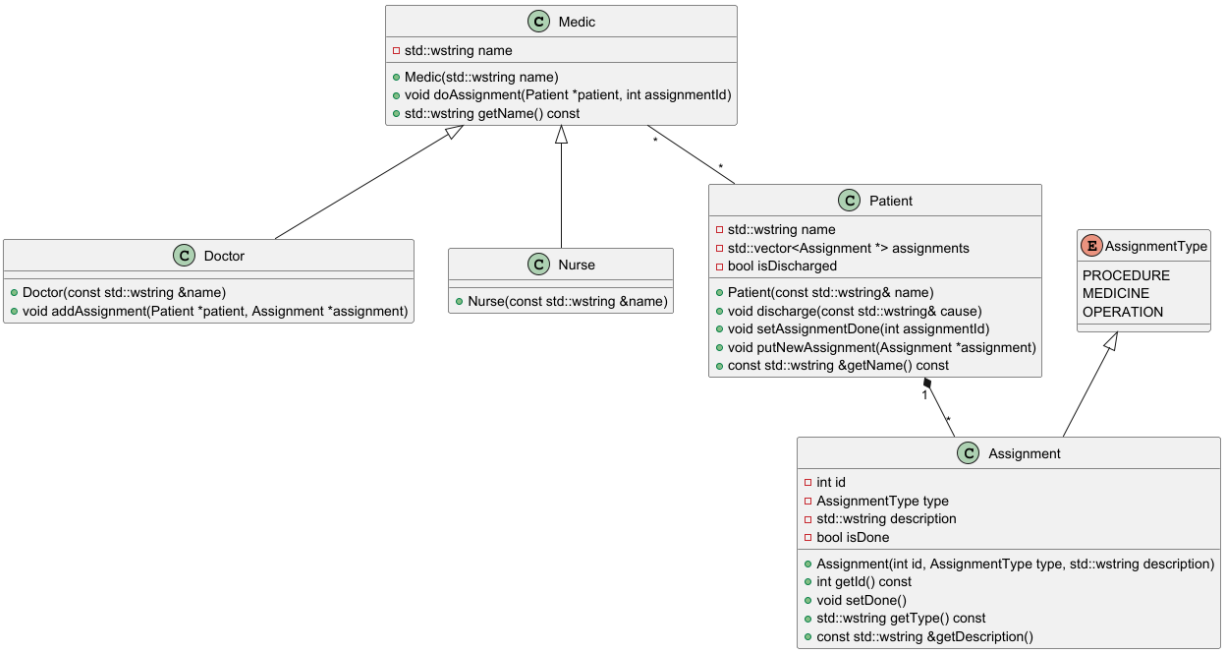

Figure 1: UML-диаграмма классов

Диаграмма последовательности:

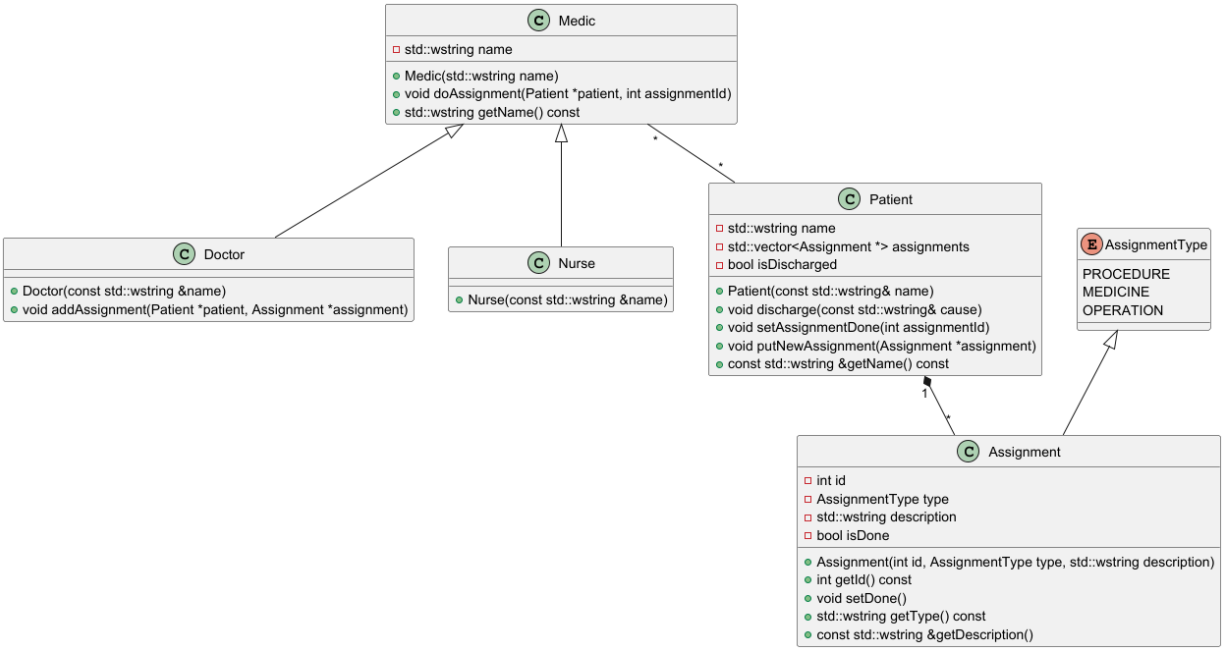

Figure 2: Диаграмма последовательности

Исходный код программы:

main.cpp

```cpp
#include "Doctor.h"
#include "Nurse.h"

int main() {
    setlocale(LC_ALL, "Russian");

    Doctor doctor1(L"Доктор №1");
    Doctor doctor2(L"Доктор №2");

    Nurse nurse(L"Медсестра №1");

    Patient patient1(L"Пациент №1");
    Patient patient2(L"Пациент №2");
    Patient patient3(L"Пациент №3");

    doctor1.addAssignment(&patient1, new Assignment(1,
        AssignmentType::PROCEDURE, L"Прогулка на свежем воздухе"));
    doctor2.addAssignment(&patient2, new Assignment(2, AssignmentType::MEDICINE,
        L"Парацетамол, 2 таблетки"));
    doctor1.addAssignment(&patient3, new Assignment(3,
        AssignmentType::OPERATION, L"Пересадка почки"));
    doctor2.addAssignment(&patient3, new Assignment(4,
        AssignmentType::PROCEDURE, L"Подтягивания, 15 раз"));

    nurse.doAssignment(&patient1, 1);
    doctor1.doAssignment(&patient2, 2);
    nurse.doAssignment(&patient3, 4);
    doctor2.doAssignment(&patient3, 3);

```

```
    patient1.discharge(L"Выздоровел");
    patient2.discharge(L"Нарушение режима");

    return 0;
}
```

Assignment.cpp

```
#include <utility>
#include "Assignment.h"

Assignment::Assignment(int id, AssignmentType type, std::wstring description) :
        id(id), type(type), description(std::move(description)), isDone(false) {}

int Assignment::getId() const {
    return id;
}

void Assignment::setDone() {
    isDone = true;
}

std::wstring Assignment::getType() const {
    switch (type) {
        case AssignmentType::PROCEDURE:
            return L"'Процедура'";
        case AssignmentType::MEDICINE:
            return L"'Лекарство'";
        case AssignmentType::OPERATION:
            return L"'Операция'";
    }
}

const std::wstring &Assignment::getDescription() {
    return description;
}
```

Doctor.cpp

```
#include <iostream>
#include "Doctor.h"

void Doctor::addAssignment(Patient *patient, Assignment *assignment) {
    log(L"Доктор " + name + L" назначает пациенту " + patient->getName() + L" процедуру
        с типом "
        + assignment->getType() + L":  " + assignment->getDescription());
    patient->putNewAssignment(assignment);
}

Doctor::Doctor(const std::wstring &name) : Medic(name) {
    log(L"Добавлен доктор " + name);
}
```

log.cpp

```
#include "log.h"

#include <iostream>

void log(const std::wstring &&message) {
```

```
    std::wcout << message << std::endl;
}
```

Medic.cpp

```
#include <iostream>
#include <utility>
#include "Medic.h"

Medic::Medic(std::wstring name) : name(std::move(name)) {}

void Medic::doAssignment(Patient *patient, int assignmentId) {
    log(L"Медицинский работник " + name + L" выполняет назначение с
        id=" + std::to_wstring(assignmentId));
    patient->setAssignmentDone(assignmentId);
}

std::wstring Medic::getName() const {
    return name;
}
```

Nurse.cpp

```
#include <iostream>
#include "Nurse.h"

Nurse::Nurse(const std::wstring &name) : Medic(name) {
    log(L"Добавлена медсестра " + name);
}
```

Patient.cpp

```
#include <iostream>
#include "Patient.h"

Patient::Patient(const std::wstring &name) : name(name), assignments(),
    isDischarged(false) {
    log(L"Добавлен пациент " + name);
}

void Patient::discharge(const std::wstring &cause) {
    log(L"Пациент " + name + L" выписан по причине:  " + cause);
    isDischarged = true;
}

void Patient::setAssignmentDone(int assignmentId) {
    for (auto &assignment: assignments) {
        if (assignment->getId() == assignmentId) {
            assignment->setDone();
            return;
        }
    }
}

void Patient::putNewAssignment(Assignment *assignment) {
    assignments.push_back(assignment);
}

const std::wstring &Patient::getName() const {
```

```
    return name;
}
```

Assignment.h

```
#ifndef LAB1_ASSIGNMENT_H
#define LAB1_ASSIGNMENT_H


#include <string>
#include "log.h"

enum AssignmentType {
    PROCEDURE,
    MEDICINE,
    OPERATION
};

class Assignment {
    int id;
    AssignmentType type;
    std::wstring description;
    bool isDone;

public:
    explicit Assignment(int id, AssignmentType type, std::wstring description);

    int getId() const;

    void setDone();

    std::wstring getType() const;

    const std::wstring &getDescription();
};


#endif //LAB1_ASSIGNMENT_H
```

Doctor.h

```
#ifndef LAB1_DOCTOR_H
#define LAB1_DOCTOR_H


#include "Medic.h"

class Doctor : public Medic {
public:
    explicit Doctor(const std::wstring &name);

    void addAssignment(Patient *patient, Assignment *assignment);
};


#endif //LAB1_DOCTOR_H
```

log.h

```cpp
#ifndef LAB1_LOG_H
#define LAB1_LOG_H


#include <iostream>

void log(const std::wstring &&message);


#endif //LAB1_LOG_H
```

Medic.h

```cpp
#ifndef LAB1_MEDIC_H
#define LAB1_MEDIC_H


#include <string>
#include "Patient.h"

class Medic {
protected:
    std::wstring name;

public:
    explicit Medic(std::wstring name);

    void doAssignment(Patient *patient, int assignmentId);

    std::wstring getName() const;
};


#endif //LAB1_MEDIC_H
```

Nurse.h

```cpp
#ifndef LAB1_NURSE_H
#define LAB1_NURSE_H


#include "Medic.h"

class Nurse : public Medic {
public:
    explicit Nurse(const std::wstring &name);
};


#endif //LAB1_NURSE_H
```

Patient.h

```cpp
#ifndef LAB1_PATIENT_H
#define LAB1_PATIENT_H


#include <string>
```

```
#include <vector>
#include "Assignment.h"

class Patient {
    std::wstring name;
    std::vector<Assignment *> assignments;
    bool isDischarged;

public:
    explicit Patient(const std::wstring &name);

    void discharge(const std::wstring &cause);

    void setAssignmentDone(int assignmentId);

    void putNewAssignment(Assignment *assignment);

    [[nodiscard]] const std::wstring &getName() const;
};


#endif //LAB1_PATIENT_H
```