МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №2

по дисциплине: ООП
тема: «Модульное программирование. Интерфейсы.»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
пр. Черников Сергей Викторович

Белгород 2024 г.

Лабораторная работа №2
«Модульное программирование. Интерфейсы.
Вариант 10

Цель работы: Получение навыков модульной декомпозиции предметной области, создания модулей. Разработка интерфейсов.

Разработать программу, состоящую из трех модулей в соответствии с указанным вариантом задания. Первый модуль – основной код программы; второй содержит интерфейсы; третий модуль – реализацию этих интерфейсов. Количество структур данных (объектов) не менее пяти.

Программа «Органайзер» (учет и планирование личного времени)

todolist.h

```
#pragma once

#include <ostream>
#include <vector>

namespace todolist {
    struct ttime {
        time_t time;
    };

    struct date {
        ttime time;
        static date now();
        date plus(ttime plusTime);

        friend std::ostream& operator<<(std::ostream& out, date& task);
    };

    enum status {
        awaits,
        in_progress,
        overdue,
        completed
    };

    // This is single function i couldn't put in .cpp file
    inline std::ostream & operator<<(std::ostream & out, todolist::status s) {
        switch (s) {
        case todolist::status::awaits: return out << "Awaits";
        case todolist::status::in_progress: return out << "In progress";
        case todolist::status::overdue: return out << "Overdue";
        case todolist::status::completed: return out << "Completed";
        default: return out << (int) s;
        }
    }

    struct task {
```

```
        private:
        static int id_counter;
        int id;
        date begin;
        date end;
        std::string description;
        bool completed;

        public:
        task(date beg, date end, std::string description);
        date get_begin();
        date get_end();
        std::string get_description();
        status get_status();
        int get_id();
        void set_destription(std::string description);
        void set_begin(date beg);
        void set_end(date end);
        void set_completed(bool completed);

        static bool task_cmp(task a, task b);

        friend std::ostream& operator<<(std::ostream& out, task& task);
    };

    struct schedule {
        public:
        void add_task(task task);
        void delete_task(int task_id);
        task get_task(int task_id);

        friend std::ostream& operator<<(std::ostream& out, schedule& task);

        private:
        std::vector<task> tasks;
    };
}
```

todolist.cpp

```
#include <algorithm>
#include <sstream>
#include <chrono>

#include "todolist.h"

namespace todolist {

    date date::now() {
        return {std::chrono::system_clock::to_time_t(std::chrono::system_clock::now())};
    }

    date date::plus(ttime plusTime) {
```

```
    return {this->time.time + plusTime.time};
}

std::ostream& operator<<(std::ostream& out, date& task) {
    std::tm * ptm = std::localtime(&task.time.time);
    char buffer[32];
    std::strftime(buffer, 32, "%a, %d.%m.%Y %H:%M:%S", ptm);

    out << std::string(buffer);
    return out;
}



int task::id_counter = 0;

task::task(date beg, date end, std::string description) {
    if (end.time.time < beg.time.time)
        throw std::invalid_argument("Completion time can't be bigger than beginning time");

    this->begin = beg;
    this->end = end;
    this->description = description;
    this->id = task::id_counter++;
    this->completed = false;
}

date task::get_begin() {
    return this->begin;
}

date task::get_end() {
    return this->end;
}

std::string task::get_description() {
    return this->description;
}

void task::set_destription(std::string description) {
    this->description = description;
}

void task::set_begin(date beg) {
    if (this->end.time.time < beg.time.time)
        throw std::invalid_argument("Completion time can't be bigger than beginning time");

    this->begin = beg;
}

void task::set_end(date end) {
    if (end.time.time < this->begin.time.time)
        throw std::invalid_argument("Completion time can't be bigger than beginning time");
```

```
    this->end = end;
}

void task::set_completed(bool completed) {
    this->completed = completed;
}

status task::get_status() {
    if (this->completed) return status::completed;

    date now_time = date::now();
    if (now_time.time.time < this->begin.time.time)
        return status::awaits;
    if (now_time.time.time < this->end.time.time)
        return status::in_progress;
    return status::overdue;
}

int task::get_id() {
    return this->id;
}

std::ostream& operator<<(std::ostream& out, task& task) {
    std::stringstream buf;
    buf << "Task:        " << task.id << "\n";
    buf << "Begin:       " << task.begin << "\n";
    buf << "End:         " << task.end << "\n";
    buf << "Description: " << task.description << "\n";
    buf << "Status:      " << task.get_status()<< "\n";

    out << buf.str();

    return out;
}

bool task::task_cmp(task a, task b) {
    return a.id == b.id;
}




void schedule::add_task(task task) {
    auto pos = std::find_if(this->tasks.begin(), this->tasks.end(), [&task](struct task item) {
        return task::task_cmp(task, item);
    });

    if (pos == this->tasks.end())
        this->tasks.push_back(task);
    else
        this->tasks[pos - this->tasks.begin()] = task;
}

void schedule::delete_task(int task_id) {
```

```
        auto pos = std::find_if(this->tasks.begin(), this->tasks.end(), [&task_id](struct task item) {
            return item.get_id() == task_id;
        });

        if (pos == this->tasks.end())
            throw std::invalid_argument("Task with such id doesn't exists");
        else
            this->tasks.erase(pos);
    }

    task schedule::get_task(int task_id) {
        auto pos = std::find_if(this->tasks.begin(), this->tasks.end(), [&task_id](struct task item) {
            return item.get_id() == task_id;
        });

        if (pos == this->tasks.end())
            throw std::invalid_argument("Task with such id doesn't exists");
        else
            return this->tasks[pos - this->tasks.begin()];
    }


    std::ostream& operator<<(std::ostream& out, schedule& schedule) {
        if (schedule.tasks.size() == 0)
            return out << "No tasks";

        out << "Tasks: \n";
        for (auto &t : schedule.tasks) {
            out << "=========================\n";
            out << t << "\n";
        }
        return out << "=========================";
    }
}
```

main.cpp

```
#include <iostream>

#include "../libs/alg/todolist/todolist.h"

int main() {
    std::cout << "Type 0 to get help\n";
    std::cout << "Type 1 to get list of tasks\n";
    std::cout << "Type 2 to create task\n";
    std::cout << "Type 3 to switch task completion\n";
    std::cout << "Type 4 to delete task\n";
    std::cout << "Type 5 to exit" << std::endl;

    todolist::schedule schedule;

    while (true) {
        int action;
```

```
        std::cin >> action;

        if (action == 0) {
            std::cout << "Type 0 to get help\n";
            std::cout << "Type 1 to get list of tasks\n";
            std::cout << "Type 2 to create task\n";
            std::cout << "Type 3 to switch task completion\n";
            std::cout << "Type 4 to delete task\n";
            std::cout << "Type 5 to exit" << std::endl;
        } else if (action == 1) {
            std::cout << "Todolist: \n";
            std::cout << schedule << std::endl;
        } else if (action == 2) {
            std::cout << "Enter description: " << std::endl;
            std::string desc;
            std::cin.ignore();
            std::getline(std::cin, desc);
            std::cout << "Enter how soon should it start (in format \"h m s\"): " << std::endl;
            long long hb = 0, mb = 0, sb = 0;
            std::cin >> hb >> mb >> sb;
            std::cout << "Enter how soon should it last (in format \"h m s\"): " << std::endl;
            long long he = 0, me = 0, se = 0;
            std::cin >> he >> me >> se;
            todolist::task task(todolist::date::now().plus({hb * 3600 + mb * 60 + sb}),
                                todolist::date::now().plus({(he + hb) * 3600 + (me + mb) * 60 + se + sb}), desc);

            try {
                schedule.add_task(task);
                std::cout << "Task successfully created" << std::endl;
            } catch (std::invalid_argument &ex) {
                std::cout << ex.what();
                std::cout << std::endl;
            }
        } else if (action == 3) {
            std::cout << "Enter task id: " << std::endl;
            int id;
            std::cin >> id;

            try {
                auto t = schedule.get_task(id);
                t.set_completed(t.get_status() != todolist::completed);
                schedule.add_task(t);
                std::cout << "Task successfully marked" << std::endl;
            } catch (std::invalid_argument &ex) {
                std::cout << ex.what();
                std::cout << std::endl;
            }
        } else if (action == 4) {
            std::cout << "Enter task id: " << std::endl;
            int id;
            std::cin >> id;

            try {
                schedule.delete_task(id);
```

```
                std::cout << "Task successfully deleted" << std::endl;
            } catch (std::invalid_argument &ex) {
                std::cout << ex.what();
                std::cout << std::endl;
            }
        } else if (action == 5)
            break;
        else
            std::cout << "Action is not recognised" << std::endl;
    }

    std::cout << "Goodnight" << std::endl;
}
```

Ссылка на репозиторий

Вывод: в ходе лабораторной работы получили навыки модульной декомпозиции предметной области, создания модулей. Разработали интерфейс. Реализовали программу.