**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МО ЭВМ**

# ОТЧЁТ

**по лабораторной работе №3**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: Логическое разделение классов**

Студент гр. 8303 _____ Парфентьев Л.М.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

**Цель работы**

Научиться организовывать логическое разделение классов.

**Задание**

Разработать и реализовать набора классов для взаимодействия пользователя с юнитами и базой. Основные требования:

- Должен быть реализован функционал управления юнитами
- Должен быть реализован функционал управления базой

**Ход выполнения работы**

- Был создан класс „медиатора“ – `Mediator`. Этот класс отвечает за взаимодействие различных объектов на карте – юнитов, нейтральных объектов, баз. В этом классе реализованы правила передвижения юнитов, атак юнитов, и т.д.
- Создан класс `Game` отвечающий за учёт баз, а также переключение ходов игроков. Класс игроков – `Player`.
- Создан класс `IostreamPlayer` – наследник `Player`, считывающий и выполняющий команды с потока ввода (например, стандартного потока ввода `std::cin`). Также этот класс печатает ответы некоторых команд на данный поток вывода (например, стандартный вывод `std::cout`).
- Имеются команды, печатающие состояние карты и базы, а также команды передвижения, атаки, использования нейтрального объекта, и создания юнита на базе.
- После каждого хода мёртвые юниты собираются (для этого написан класс `ZombieCollector`, отслеживающий события смерти юнитов).

**Выводы**

В ходе выполнения лабораторной работы было изучено логическое разделение классов.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: mediator.hpp

```
#ifndef _H_MEDIATOR_HPP
#define _H_MEDIATOR_HPP

#include "point.hpp"
#include "map.hpp"
#include "unit.hpp"


class Mediator {
    Map *_map;

public:
    Mediator(Map *map)
        :_map{map} {}

    MapInfo infoAt(Vec2 pt);

    Vec2 mapSize();

    bool moveUnitTo(Unit *u, Vec2 to);

    bool attackTo(Unit *u, Vec2 to);

    bool useObject(Unit *u);

    bool spawnUnit(Unit *u, Vec2 at);
    bool teleportUnit(Unit *u, Vec2 to);
};


#endif
```

Название файла: player.hpp

```
#ifndef _H_PLAYER_HPP
#define _H_PLAYER_HPP

#include <string>
#include <utility>

#include "mediator.hpp"
#include "base.hpp"


class Player {
```

```
    std::string _name;

public:
    Player(std::string name)
        :_name{std::move(name)} {}

    const std::string &name() const { return _name; }

    virtual bool takeTurn(Mediator *m, Base *b) =0;

    virtual ~Player() {}
};

#endif
```

Название файла: zombie_collector.cpp

Название файла: game.hpp

```
#ifndef _H_GAME_HPP
#define _H_GAME_HPP

#include <vector>

#include "event.hpp"
#include "map.hpp"
#include "base.hpp"
#include "player.hpp"
#include "zombie_collector.hpp"
#include "mediator.hpp"


class Game: public EventForwarder {
    Map *_map;
    Mediator *_med;

    struct BaseRecord {
        Base *base;
        Player *player;
    };

    std::vector<BaseRecord> _recs {};
    int _next = 0;
    int _players = 0;

    ZombieCollector *_coll;

public:
    Game(Map *map)
        :_map{map},
         _med{new Mediator {map}},
         _coll{new ZombieCollector {}} {}
```

```
    int addBase(Base *b);
    void setPlayer(int base_idx, Player *p);

    int basesCount() const;
    int playersCount() const;

    Base *baseByIdx(int idx) const;

    void spin();

    ~Game();
};


#endif
```

Название файла: game.cpp

```
#include "base.hpp"
#include "game.hpp"
#include "event.hpp"
#include "event_types.hpp"


int
Game::addBase(Base *base)
{
    base->subscribe(_coll);
    base->subscribe(this);

    _recs.push_back(BaseRecord{base, nullptr});

    emit(new events::BaseAdded {base});

    return (int)(_recs.size() - 1);
}

void
Game::setPlayer(int base_idx, Player *p)
{
    Player *&p_place = _recs[base_idx].player;
    if (p_place) {
        delete p_place;
        --_players;
    }
    p_place = p;
    if (p) {
        ++_players;
    }
}
```

```
int
Game::basesCount() const
{
    return _recs.size();
}

int
Game::playersCount() const
{
    return _players;
}

Base *
Game::baseByIdx(int idx) const
{
    return _recs[idx].base;
}

void
Game::spin()
{
    auto &rec = _recs[_next];
    Player *p = rec.player;
    Base *b = rec.base;

    if (p) {
        emit(new events::TurnStarted {p});

        bool res = p->takeTurn(_med, b);

        _coll->collect(_map);

        emit(new events::TurnOver {p});

        if (!res) {
            setPlayer(_next, nullptr);
        }
    }

    if (++_next == (int)_recs.size()) {
        _next = 0;
    }
}

Game::~Game()
{
    for (int i = 0; i < (int)_recs.size(); ++i) {
        setPlayer(i, nullptr);
        auto &rec = _recs[i];
        rec.base->unsubscribe(this);
        rec.base->unsubscribe(_coll);
    }
```

```
    delete _coll;
    delete _map;
}
```

Название файла: iostream_player.hpp

```
#ifndef _H_STDIO_PLAYER_HPP
#define _H_STDIO_PLAYER_HPP

#include <iostream>
#include <string>
#include <map>
#include <queue>

#include "point.hpp"
#include "game.hpp"

#include "object_print.hpp"


class IostreamPlayer;

class IostreamCommand {
public:
    // -> whether to end turn
    virtual bool execute(IostreamPlayer *p, Mediator *m, Base *b) =0;

    virtual ~IostreamCommand() {}
};

class IostreamPlayer: public Player {
    std::ostream *_os = nullptr;
    std::istream *_is = nullptr;
    bool _free_os, _free_is;

    std::map<std::string, IostreamCommand *> _cmd_tab {};

    void
    addCommand(const std::string &str,
               IostreamCommand *cmd);

public:
    IostreamPlayer(std::string name);

    void
    setOstream(std::ostream *os) { _os = os; _free_is = true; }
    void
    setOstream(std::ostream &os) { _os = &os; _free_os = false; }

    std::ostream &
    ostream() const { return *_os; }
```

```
    void
    setIstream(std::istream *is) { _is = is; _free_is = true; }
    void
    setIstream(std::istream &is) { _is = &is; _free_is = false; }

    std::istream &
    istream() const { return *_is; }

    virtual bool
    takeTurn(Mediator *m, Base *b) override;

    int
    readInt();

    Unit *
    readUnitId(Base *b);

    Vec2
    readVec2();

    std::string
    readString();
};

namespace iostream_commands {

    class Move: public IostreamCommand {
    public:
        virtual bool
        execute(IostreamPlayer *p, Mediator *m, Base *b) override
        {
            Unit *u = p->readUnitId(b);
            Vec2 to = p->readVec2();
            if (!u)
                return false;

            if (!m->moveUnitTo(u, to)) {
                p->ostream() << "Can't move unit " << u
                             << " to " << to << "\n";
                return false;
            }

            return true;
        }
    };

    class Attack: public IostreamCommand {
    public:
        virtual bool
        execute(IostreamPlayer *p, Mediator *m, Base *b) override
        {
```

```
        Unit *u = p->readUnitId(b);
        Vec2 to = p->readVec2();
        if (!u)
            return false;

        if (!m->attackTo(u, to)) {
            p->ostream() << "Unit " << u
                         << " can't attack to " << to << "\n";
            return false;
        }

        return true;
    }
};

class Use: public IostreamCommand {
public:
    virtual bool
    execute(IostreamPlayer *p, Mediator *m, Base *b) override
    {
        Unit *u = p->readUnitId(b);
        if (!u)
            return false;

        if (!m->useObject(u)) {
            p->ostream() << "Unit " << u
                         << " can't use any object there\n";
            return false;
        }

        return true;
    }
};

class Create: public IostreamCommand {
public:
    virtual bool
    execute(IostreamPlayer *p, Mediator *m, Base *b) override
    {
        std::string s = p->readString();

        if (!b->canCreateUnit(s)) {
            p->ostream() << "Can't create unit of type "
                         << s << "\n";
            return false;
        }

        int id = b->createUnit(s, m);
        if (id < 0) {
            p->ostream() << "Failed to create a unit of type "
                         << s << "\n";
            return false;
```

```
        }

        p->ostream() << "New unit of type " << s
                     << ": " << id << "\n";
        return true;
    }
};

class FindBase: public IostreamCommand {
public:
    virtual bool
    execute(IostreamPlayer *p, Mediator *, Base *b) override
    {
        p->ostream() << "Base: " << b << "\n";
        return false;
    }
};

class ListUnits: public IostreamCommand {
public:
    virtual bool
    execute(IostreamPlayer *p, Mediator *, Base *b) override
    {
        p->ostream() << "Units:";
        for (auto iter = b->unitsBegin();
             iter != b->unitsEnd();
             ++iter) {
            p->ostream() << "- " << iter.id()
                         << ": " << iter.unit() << "\n";
        }

        p->ostream() << std::endl;
        return false;
    }
};

class DescribeAt: public IostreamCommand {
public:
    virtual bool
    execute(IostreamPlayer *p, Mediator *m, Base *) override
    {
        auto pos = p->readVec2();
        auto info = m->infoAt(pos);

        p->ostream() << "At " << pos << "\n";
        p->ostream()
            << "- Landscape: " << info.landscape() << "\n";

        if (auto *b = info.base()) {
            p->ostream() << "- Base: " << b << "\n";
        }
```

```
            if (auto *u = info.unit()) {
                p->ostream() << "- Unit: " << u << "\n";
            }

            if (auto *n = info.neutralObject()) {
                p->ostream() << "- Object: " << n << "\n";
            }

            p->ostream() << std::endl;
            return false;
        }
    };

    class PrintMap: public IostreamCommand {
    public:
        virtual bool
        execute(IostreamPlayer *p, Mediator *m, Base *) override
        {
            auto from = p->readVec2();
            auto to = p->readVec2();

            p->ostream() << "From " << from
                         << " to " << to << ":\n";

            for (int y = from.y(); y < to.y(); ++y) {
                for (int x = from.x(); x < to.x(); ++x) {
                    if (x != from.x()) {
                        p->ostream() << " ";
                    }
                    displayMapInfo(p->ostream(), m->infoAt({x, y}));
                }
                p->ostream() << "\n";
            }

            p->ostream() << std::endl;
            return false;
        }
    };

    class Skip: public IostreamCommand {
    public:
        virtual bool
        execute(IostreamPlayer *, Mediator *, Base *) override
        {
            return true;
        }
    };

}

#endif
```

Название файла: iostream_player.cpp

```
#include <iostream>
#include <string>

#include "game.hpp"
#include "base.hpp"
#include "iostream_player.hpp"


void
IostreamPlayer::addCommand(const std::string &str,
                           IostreamCommand *cmd)
{
    _cmd_tab[str] = cmd;
}

IostreamPlayer::IostreamPlayer(std::string name)
    :Player{name}
{
    addCommand("move", new iostream_commands::Move {});
    addCommand("attack", new iostream_commands::Attack {});
    addCommand("use", new iostream_commands::Use {});
    addCommand("create", new iostream_commands::Create {});
    addCommand("base", new iostream_commands::FindBase {});
    addCommand("units", new iostream_commands::ListUnits {});
    addCommand("describe", new iostream_commands::DescribeAt {});
    addCommand("map", new iostream_commands::PrintMap {});
    addCommand("skip", new iostream_commands::Skip {});
}

bool
IostreamPlayer::takeTurn(Mediator *m, Base *b)
{
    for (;;) {
        std::string cmd_name;
        (*_is) >> cmd_name;

        if (_is->fail()) {
            return false;
        }

        auto iter = _cmd_tab.find(cmd_name);
        if (iter == _cmd_tab.end()) {
            (*_os) << "Unknown command: \"" << cmd_name << "\"\n";
            continue;
        }

        if (iter->second->execute(this, m, b)) {
            break;
        }
    }
```

```
    return true;
}

int
IostreamPlayer::readInt()
{
    int x;
    (*_is) >> x;
    return x;
}

Unit *
IostreamPlayer::readUnitId(Base *b)
{
    int id = readInt();
    Unit *u = b->getUnitById(id);
    if (!u) {
        (*_os) << "No such unit: " << id << "\n";
    }

    return u;
}

Vec2
IostreamPlayer::readVec2()
{
    int x, y;
    (*_is) >> x >> y;
    return {x, y};
}

std::string
IostreamPlayer::readString()
{
    std::string s;
    (*_is) >> s;
    return s;
}
```