Лабораторная работа №6
«ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ»
(часть 2)

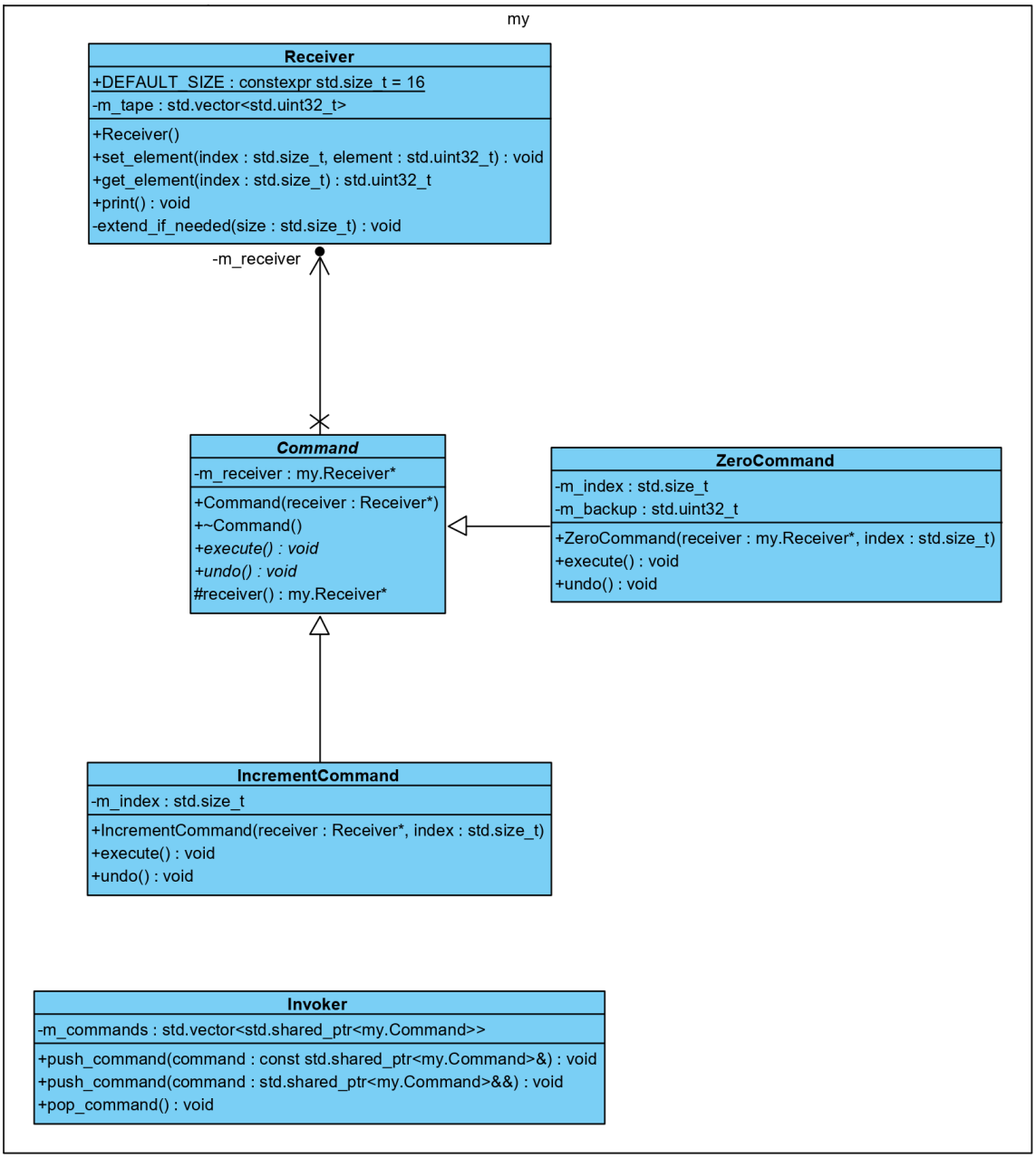
по дисциплине "Методы программирования"

Выполнил Ковалев Даниил
СКБ171, вариант 2 (12)
МИЭМ НИУ ВШЭ

Условие. Используя паттерн Command (команда) разработать тестовый язык (например, типа shell), поддерживающий несколько видов команд с возможностью их отмены (undo).

Был разработан язык, который может выполнять операции над лентой с целыми неотрицательными числами: инкремент и обнуление числа на заданной позиции.

UML-диаграмма классов:



Структура проекта:

lab6:
- CMakeLists.txt
- command.h — файл с описанием необходимых команд
- command.cpp — файл с реализацией необходимых команд
- invoker.h — файл с описанием класса, вызывающего команды
- invoker.cpp — файл с реализацией класса, вызывающего команды
- receiver.h — файл с описанием класса, обрабатывающего команды (лента с числами)
- receiver.cpp — файл с реализацией класса, обрабатывающего команды (лента с числами)
- main.cpp — файл с примером использования паттерна Command

CMakeLists.txt — см. ЛР1

Результат работы программы:

```
Increment element number 5...
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
Increment element number 6...
0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
Undo increment element number 6...
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
Increment element number 5...
0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0
Increment element number 7...
0 0 0 0 0 2 0 1 0 0 0 0 0 0 0 0
Zero element number 5...
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
Undo zero element number 5...
0 0 0 0 0 2 0 1 0 0 0 0 0 0 0 0
Undo increment element number 7...
0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0
Undo increment element number 5...
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
```

Листинги с исходным кодом всех файлов расположены на следующих страницах отчета.

Листинг 1: lab6/CMakeLists.txt

```
set(PROJECT_NAME 6_patterns)

project(${PROJECT_NAME} LANGUAGES CXX)

set(SOURCES main.cpp command.cpp receiver.cpp invoker.cpp)
set(HEADERS command.h receiver.h invoker.h)

add_executable(${PROJECT_NAME} ${SOURCES} ${HEADERS})
```

Листинг 2: lab6/command.h

```
#ifndef COMMAND_H
#define COMMAND_H

#include "receiver.h"

namespace my
{

class Command
{
public:
    explicit Command(Receiver* receiver);
    virtual ~Command() = default;
    virtual void execute() const = 0;
    virtual void undo() const = 0;

protected:
    Receiver* receiver() const;

private:
    Receiver* m_receiver;
};


class IncrementCommand : public Command
{
public:
    IncrementCommand(Receiver* receiver, std::size_t index);
    void execute() const override;
    void undo() const override;

private:
    std::size_t m_index;
};

class ZeroCommand : public Command
{
public:
    ZeroCommand(Receiver* receiver, std::size_t index);
    void execute() const override;
    void undo() const override;

private:
    std::size_t m_index;
    mutable std::uint32_t m_backup;
};

} // namespace my

#endif // COMMAND_H
```

Листинг 3: lab6/command.cpp

```cpp
#include "command.h"
#include "receiver.h"

#include <stdexcept>
#include <iostream>

namespace my
{

Receiver* Command::receiver() const
{
    return m_receiver;
}

Command::Command(Receiver* receiver)
    : m_receiver(receiver)
{
    if (m_receiver == nullptr)
        throw std::invalid_argument("Receiver can not be null!");
}


IncrementCommand::IncrementCommand(Receiver* receiver, std::size_t index)
    : Command(receiver)
    , m_index(index)
{
}

void IncrementCommand::execute() const
{
    std::cerr << "Increment element number " << m_index << "...\n";
    receiver()->set_element(m_index, receiver()->get_element(m_index) + 1);
}

void IncrementCommand::undo() const
{
    std::cerr << "Undo increment element number " << m_index << "...\n";
    receiver()->set_element(m_index, receiver()->get_element(m_index) - 1);
}


ZeroCommand::ZeroCommand(Receiver* receiver, std::size_t index)
    : Command(receiver)
    , m_index(index)
    , m_backup(this->receiver()->get_element(m_index))
{
}

void ZeroCommand::execute() const
{
    std::cerr << "Zero element number " << m_index << "...\n";
    m_backup = receiver()->get_element(m_index);
    receiver()->set_element(m_index, 0);
}

void ZeroCommand::undo() const
{
    std::cerr << "Undo zero element number " << m_index << "...\n";
    receiver()->set_element(m_index, m_backup);
}

} // namespace my
```

Листинг 4: lab6/invoker.h

```h
#ifndef INVOKER_H
#define INVOKER_H

#include "command.h"

#include <vector>
#include <memory>

namespace my
{

class Invoker
{

private:
    std::vector<std::shared_ptr<Command>> m_commands;

public:
    void push_command(const std::shared_ptr<Command>& command);
    void push_command(std::shared_ptr<Command>&& command);
    void pop_command();
};

} // namespace my

#endif // INVOKER_H
```

Листинг 5: lab6/invoker.cpp

```cpp
#include "invoker.h"

namespace my
{

void Invoker::push_command(const std::shared_ptr<Command>& command)
{
    m_commands.push_back(command);
    m_commands.back()->execute();
}

void Invoker::push_command(std::shared_ptr<Command>&& command)
{
    m_commands.push_back(std::move(command));
    m_commands.back()->execute();
}

void Invoker::pop_command()
{
    m_commands.back()->undo();
    m_commands.pop_back();
}

} // namespace my
```

Листинг 6: lab6/receiver.h

```h
#ifndef RECEIVER_H
#define RECEIVER_H

#include <vector>
#include <cstdint>

namespace my
{

```

```
10 class Receiver
11 {
12 public:
13     static inline constexpr std::size_t DEFAULT_SIZE = 16;
14
15     Receiver();
16     void set_element(std::size_t index, std::uint32_t element);
17     std::uint32_t get_element(std::size_t index) const;
18     void print() const;
19
20 private:
21     std::vector<std::uint32_t> m_tape;
22
23     void extend_if_needed(std::size_t size);
24 };
25
26 } // namespace my
27
28 #endif // RECEIVER_H
```

Листинг 7: lab6/receiver.cpp

```
1  #include "receiver.h"
2
3  #include <vector>
4  #include <cstdint>
5  #include <iostream>
6
7  namespace my
8  {
9
10 void Receiver::extend_if_needed(std::size_t size)
11 {
12     if (m_tape.size() < size)
13         m_tape.resize(size, 0);
14 }
15
16 Receiver::Receiver()
17     : m_tape(DEFAULT_SIZE, 0)
18 {
19 }
20
21 void Receiver::set_element(std::size_t index, std::uint32_t element)
22 {
23     extend_if_needed(index + 1);
24     m_tape[index] = element;
25 }
26
27 std::uint32_t Receiver::get_element(std::size_t index) const
28 {
29     if (index < m_tape.size())
30         return m_tape[index];
31     return 0;
32 }
33
34 void Receiver::print() const
35 {
36     for (std::uint32_t element : m_tape)
37         std::cout << element << ' ';
38     std::cout << '\n';
39 }
40
41 } // namespace my
```

Листинг 8: lab6/main.cpp

```cpp
#include "invoker.h"
#include "command.h"
#include "receiver.h"

#include <iostream>
#include <vector>
#include <memory>
#include <stdexcept>

int main() try
{
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    std::unique_ptr<my::Receiver> receiver = std::make_unique<my::Receiver>();
    my::Invoker invoker;

    std::vector<std::shared_ptr<my::Command>> increments(my::Receiver::DEFAULT_SIZE);
    for (std::size_t i = 0; i < increments.size(); ++i)
        increments[i] = std::make_shared<my::IncrementCommand>(receiver.get(), i);

    std::vector<std::shared_ptr<my::Command>> zeros(my::Receiver::DEFAULT_SIZE);
    for (std::size_t i = 0; i < zeros.size(); ++i)
        zeros[i] = std::make_shared<my::ZeroCommand>(receiver.get(), i);

    invoker.push_command(increments[5]);
    receiver->print();
    invoker.push_command(increments[6]);
    receiver->print();
    invoker.pop_command();
    receiver->print();
    invoker.push_command(increments[5]);
    receiver->print();
    invoker.push_command(increments[7]);
    receiver->print();
    invoker.push_command(zeros[5]);
    receiver->print();
    invoker.pop_command();
    receiver->print();
    invoker.pop_command();
    receiver->print();
    invoker.pop_command();
    receiver->print();
}
catch (const std::exception& e)
{
    std::cerr << e.what() << '\n';
    return 1;
}
```