**Лабораторная работа № 4**

**Рефакторинг программного кода. Упрощение вызовов методов**

**Цель работы**

Исследовать эффективность рефакторинга программного кода за счет упрощения вызовов методов. Получить практические навыки упрощения вызовов методов при рефакторинге объектно-ориентированных программ.

**Постановка задачи**

1. Выбрать фрагмент программного кода для рефакторинга.

2. Выполнить рефакторинг программного кода, применив не менее 7 приемов, рассмотренных в разделе 2.2.

3. Составить отчет, содержащий подробное описание каждого модифицированн

**Ход работы**

1. Переименование метода (Rename Method)

Код до рефакторинга:

**void Studying::ChToContStud()**

{

if (AverageMark > 90)

cout << "Student has a high chance that he will continue his studies, moreover, he may receive an increased scholarship" << endl;

else if ((AverageMark < 90) && (AverageMark >= 75))

cout << "Student has a chance that he will continue his studies" << endl;

else if (AverageMark < 75)

cout << "Student has no chances" << endl;

}

Код после рефакторинга:

**void Studying::EvaluateContinuationChances()**

{

if (AverageMark > 90)

cout << "Student has a high chance that he will continue his studies, moreover, he may receive an increased scholarship" << endl;

else if ((AverageMark < 90) && (AverageMark >= 75))

cout << "Student has a chance that he will continue his studies" << endl;

else if (AverageMark < 75)

cout << "Student has no chances" << endl;

}

2. Добавление параметра (Add Parameter)

Код до рефакторинга:

void printStudentsAverageGrades() const

{

for (const auto &student : students\_)

{

std::cout << student.getName() << ": ";

double mathAverage = student.getAverageGrade("Math");

double physicsAverage = student.getAverageGrade("Physics");

std::cout << "Math: " << mathAverage << ", Physics: " << physicsAverage << std::endl;

}

}

Код после рефакторинга:

void printStudentsAverageGrades(const string &subject = "") const

{

for (const auto &student : students\_)

{

double subjectAverage = student.getAverageGrade(subject);

if (subject.empty() || subjectAverage != -1)

{

cout << student.getName() << ": ";

double mathAverage = student.getAverageGrade("Math");

double physicsAverage = student.getAverageGrade("Physics");

cout << "Math: " << mathAverage << ", Physics: " << physicsAverage;

if (!subject.empty())

{

cout << ", " << subject << ": " << subjectAverage;

}

cout << endl;

}

}

}

3. Удаление параметра (Remove Parameter)

Код до рефакторинга:

Код после рефакторинга:

4. Разделение запроса и модификатора (Separate Query from Modifier)

Код до рефакторинга:

void processTransactions(const std::vector<std::pair<int, std::string>> &transactions)

{

for (const auto &transaction : transactions)

{

**BankAccount \*account = findAccount(transaction.first);**

if (account != nullptr)

{

std::string operation = transaction.second.substr(0, 1);

double amount = std::stod(transaction.second.substr(2));

**if (operation == "D")**

**{**

**account->deposit(amount);**

**}**

**else if (operation == "W")**

**{**

**account->withdraw(amount);**

**}**

**else**

**{**

**std::cout << "Invalid transaction type.\n";**

**}**

**}**

**else**

**{**

**std::cout << "Account not found.\n";**

}

}

}

Код после рефакторинга:

**std::vector<BankAccount\*> getAccounts() {**

**std::vector<BankAccount\*> accountsPtr;**

**for (auto& account : accounts\_) {**

**accountsPtr.push\_back(&account);**

**}**

**return accountsPtr;**

**}**

static void processTransaction(BankAccount\* account, const std::string& transaction) {

**if (account != nullptr) {**

**std::string operation = transaction.substr(0, 1);**

**double amount = std::stod(transaction.substr(2));**

**if (operation == "D") {**

**account->deposit(amount);**

**} else if (operation == "W") {**

**account->withdraw(amount);**

**} else {**

**std::cout << "Invalid transaction type.\n";**

**}**

**} else {**

**std::cout << "Account not found.\n";**

}

}

5. Параметризация метода (Parameterize Method)

Код до рефакторинга:

void processIntegers(std::vector<int> data)

{

int sum = 0;

for (int num : data)

{

sum += num;

}

std::cout << "Сумма целых чисел: " << sum << std::endl;

}

void processDoubles(std::vector<double> data)

{

double product = 1.0;

for (double num : data)

{

product \*= num;

}

std::cout << "Произведение дробных чисел: " << product << std::endl;

}

void processStrings(std::vector<std::string> data)

{

std::string concatenatedString;

for (const std::string &str : data)

{

concatenatedString += str + " ";

}

std::cout << "Объединенная строка: " << concatenatedString << std::endl;

}

Код после рефакторинга:

template <typename T>

T processData(std::vector<T> data)

{

T result = data[0];

for (size\_t i = 1; i < data.size(); ++i)

{

result += data[i];

}

return result;

}

6. Замена параметра явными методами (Replace Parameter with Explicit Methods)

Код до рефакторинга:

#include <iostream>

#include <fstream>

#include <string>

enum FileType

{

TXT,

CSV,

JSON

};

class FileProcessor

{

public:

void processFile(const std::string &filename, FileType fileType)

{

std::ifstream file(filename);

if (!file.is\_open())

{

throw std::runtime\_error("Failed to open file");

}

std::string line;

**while (std::getline(file, line))**

**{**

**switch (fileType)**

**{**

**case TXT:**

**processTxtLine(line);**

**break;**

**case CSV:**

**processCsvLine(line);**

**break;**

**case JSON:**

**processJsonLine(line);**

**break;**

**default:**

**throw std::invalid\_argument("Invalid file type");**

**}**

**}**

file.close();

}

void processTxtLine(const std::string &line)

{

std::cout << "Processing TXT line: " << line << std::endl;

}

void processCsvLine(const std::string &line)

{

std::cout << "Processing CSV line: " << line << std::endl;

}

void processJsonLine(const std::string &line)

{

std::cout << "Processing JSON line: " << line << std::endl;

}

};

int main()

{

FileProcessor processor;

processor.processFile("file.txt", TXT);

processor.processFile("file.csv", CSV);

processor.processFile("file.json", JSON);

return 0;

}

Код после рефакторинга:

#include <iostream>

#include <fstream>

#include <string>

class FileProcessor

{

**public:**

**void processTxtFile(const std::string &filename)**

**{**

**processFile(filename, &FileProcessor::processTxtLine);**

**}**

**void processCsvFile(const std::string &filename)**

**{**

**processFile(filename, &FileProcessor::processCsvLine);**

**}**

**void processJsonFile(const std::string &filename)**

**{**

**processFile(filename, &FileProcessor::processJsonLine);**

**}**

void processFile(const std::string &filename, void (FileProcessor::\*processLine)(const std::string &))

{

std::ifstream file(filename);

if (!file.is\_open())

{

throw std::runtime\_error("Failed to open file");

}

std::string line;

while (std::getline(file, line))

{

(this->\*processLine)(line);

}

file.close();

}

void processTxtLine(const std::string &line)

{

std::cout << "Processing TXT line: " << line << std::endl;

}

void processCsvLine(const std::string &line)

{

std::cout << "Processing CSV line: " << line << std::endl;

}

void processJsonLine(const std::string &line)

{

std::cout << "Processing JSON line: " << line << std::endl;

}

};

int main()

{

FileProcessor processor;

processor.processTxtFile("file.txt");

processor.processCsvFile("file.csv");

processor.processJsonFile("file.json");

return 0;

}

7. Сохранение всего объекта (Preserve Whole Object)

Код до рефакторинга:

Код после рефакторинга:

8. Замена параметра вызовом метода (Replace Parameter with Method)

Код до рефакторинга:

Код после рефакторинга:

9. Введение граничного объекта (Introduce Parameter Object)

Код до рефакторинга:

Код после рефакторинга:

10. Удаление метода установки значения (Remove Setting Method)

Код до рефакторинга:

class Student

{

public:

Student(const std::string &name, int age, double gpa)

: name\_(name), age\_(age), gpa\_(gpa) {}

**void setName(const std::string &name)**

**{**

**name\_ = name;**

**}**

const std::string &getName() const

{

return name\_;

}

После рефакторинга был удалён метод setName, посколько имя студента не должно меняться после создания.

11. Сокрытие метода (Hide Method)

Код до рефакторинга:

#include <iostream>

#include <fstream>

#include <string>

enum FileType

{

TXT,

CSV,

JSON

};

class FileProcessor

{

**public:**

void processFile(const std::string &filename, FileType fileType)

{

std::ifstream file(filename);

if (!file.is\_open())

{

throw std::runtime\_error("Failed to open file");

}

std::string line;

while (std::getline(file, line))

{

switch (fileType)

{

case TXT:

processTxtLine(line);

break;

case CSV:

processCsvLine(line);

break;

case JSON:

processJsonLine(line);

break;

default:

throw std::invalid\_argument("Invalid file type");

}

}

file.close();

}

void processTxtLine(const std::string &line)

{

std::cout << "Processing TXT line: " << line << std::endl;

}

void processCsvLine(const std::string &line)

{

std::cout << "Processing CSV line: " << line << std::endl;

}

void processJsonLine(const std::string &line)

{

std::cout << "Processing JSON line: " << line << std::endl;

}

};

int main()

{

FileProcessor processor;

processor.processFile("file.txt", TXT);

processor.processFile("file.csv", CSV);

processor.processFile("file.json", JSON);

return 0;

}

Код после рефакторинга:

#include <iostream>

#include <fstream>

#include <string>

class FileProcessor

{

**public:**

void processTxtFile(const std::string &filename)

{

processFile(filename, &FileProcessor::processTxtLine);

}

void processCsvFile(const std::string &filename)

{

processFile(filename, &FileProcessor::processCsvLine);

}

void processJsonFile(const std::string &filename)

{

processFile(filename, &FileProcessor::processJsonLine);

}

**private:**

void processFile(const std::string &filename, void (FileProcessor::\*processLine)(const std::string &))

{

std::ifstream file(filename);

if (!file.is\_open())

{

throw std::runtime\_error("Failed to open file");

}

std::string line;

while (std::getline(file, line))

{

(this->\*processLine)(line);

}

file.close();

}

void processTxtLine(const std::string &line)

{

std::cout << "Processing TXT line: " << line << std::endl;

}

void processCsvLine(const std::string &line)

{

std::cout << "Processing CSV line: " << line << std::endl;

}

void processJsonLine(const std::string &line)

{

std::cout << "Processing JSON line: " << line << std::endl;

}

};

int main()

{

FileProcessor processor;

processor.processTxtFile("file.txt");

processor.processCsvFile("file.csv");

processor.processJsonFile("file.json");

return 0;

}

12. Замена конструктора фабричным методом (Replace Constructor with Factory Method)

Код до рефакторинга:

class BankAccount

{

public:

BankAccount(const std::string &owner, double balance, int account\_number)

: owner\_(owner), balance\_(balance), account\_number\_(account\_number)

{

if (owner.empty())

{

throw std::invalid\_argument("Owner's name cannot be empty");

}

if (balance < 0)

{

throw std::invalid\_argument("Balance cannot be negative");

}

if (account\_number <= 0)

{

throw std::invalid\_argument("Account number must be positive");

}

}

Код после рефакторинга:

class BankAccountFactory

{

public:

static std::unique\_ptr<BankAccount> create(const std::string &owner, double balance, int account\_number)

{

if (owner.empty())

{

throw std::invalid\_argument("Owner's name cannot be empty");

}

if (balance < 0)

{

throw std::invalid\_argument("Balance cannot be negative");

}

if (account\_number <= 0)

{

throw std::invalid\_argument("Account number must be positive");

}

return std::make\_unique<BankAccount>(owner, balance, account\_number);

}

};

13. Инкапсуляция нисходящего преобразования типа (Encapsulate Downcast)

Код до рефакторинга:

Код после рефакторинга:

14. Замена кода ошибки исключительной ситуацией (Replace Error Code with Exception)

Код до рефакторинга:

std::string readFile(const std::string &filename)

{

std::ifstream file(filename);

**if (!file.is\_open())**

**{**

**return "";**

**}**

std::string content((std::istreambuf\_iterator<char>(file)), std::istreambuf\_iterator<char>());

return content;

}

Код после рефакторинга:

std::string readFile(const std::string &filename)

{

std::ifstream file(filename);

**if (!file.is\_open())**

**{**

**throw std::runtime\_error("Error: File not found.");**

**}**

std::string content((std::istreambuf\_iterator<char>(file)), std::istreambuf\_iterator<char>());

return content;

}

15. Замена исключительной ситуации проверкой (Replace Exception with Test)

Код до рефакторинга:

void open(const std::string &filename)

{

std::ifstream file(filename);

if (!file.is\_open())

{

throw std::runtime\_error("Could not open file");

}

std::string line;

while (std::getline(file, line))

{

buffer\_.push\_back(line);

}

}

Код после рефакторинга:

bool open(const std::string &filename)

{

std::ifstream file(filename);

if (!file.is\_open())

{

return false;

}

std::string line;

buffer\_.clear(); // clear previous content

while (std::getline(file, line))

{

buffer\_.push\_back(line);

}

return true;

}

**ВЫВОДЫ**

В ходе выполнения лабораторной работы были изучены методы рефакторинга программного кода за счет упрощения вызовов методов.

Были выбраны участки кода, нуждающиеся в рефакторинге данными методами и отрефакторены, в результате получившийся код позволил упростить вызов методов, стало проще понять принципы обращения к этим методам, а также улучшилась читабельность методов.