Технологии конструирования программного обеспечения

Отчет по лабораторной работе № 02

Группа: 221-329 **Студент:** Минчаков Аркадий Сергеевич

Задание на лабораторную работу:

Для заданного варианта задания разработать UML-диаграмму классов и диаграмму последовательности. Разработать консольное приложение (C++, C#). Допускается вводить дополнительные понятия предметной области. В программе предусмотреть тестирование функциональности созданных объектов классов.

Паттерн Builder

Имеется текст статьи в формате TXT. Статья состоит из заголовка (первая строка), фамилий авторов (вторая строка), самого текста статьи и хеш-кода текста статьи (последняя строка). Написать приложение, позволяющее конвертировать документ в формате TXT в документ формата XML. Необходимо также проверять корректность хеш-кода статьи.

Паттерн Abstract Factory

Разработать систему Кинопрокат. Пользователь может выбрать определённую киноленту, при заказе киноленты указывается язык звуковой дорожки, который совпадает с языком файла субтитров. Система должна поставлять фильм с требуемыми характеристиками, причём при смене языка звуковой дорожки должен меняться и язык файла субтитров и наоборот.

Паттерн Factory Method

Фигуры игры «тетрис». Реализовать процесс случайного выбора фигуры из конечного набора фигур. Предусмотреть появление супер-фигур с большим числом клеток, чем обычные.

UML-диаграмма классов:

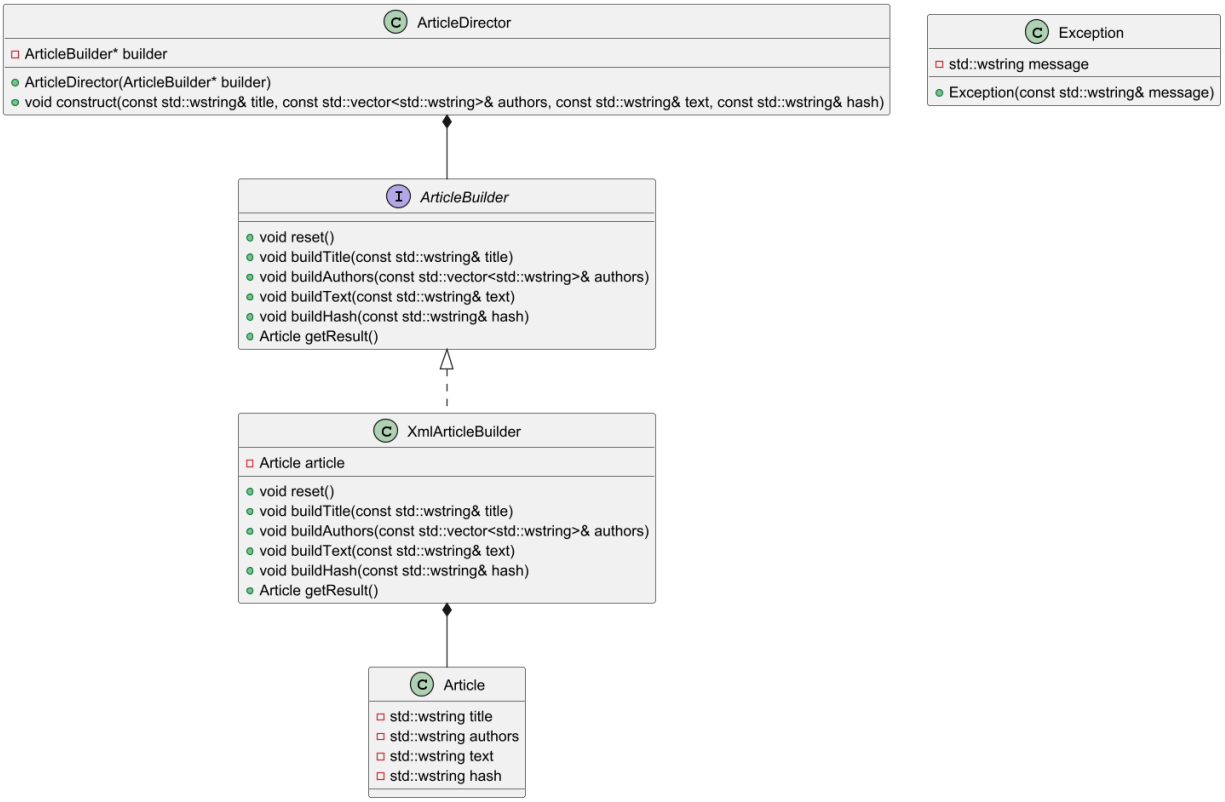

Figure 1: UML-диаграмма классов

Диаграмма последовательности:

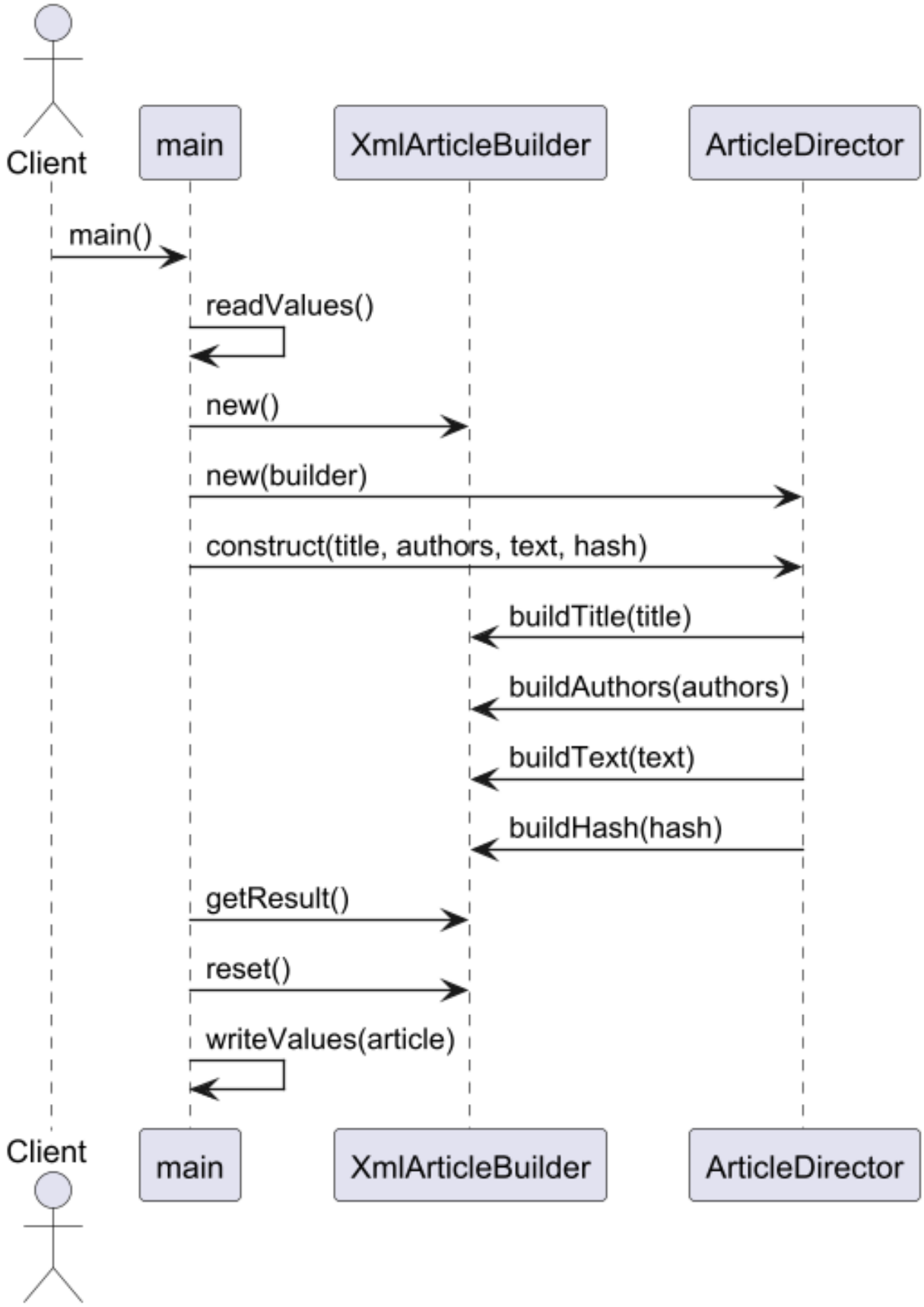

Figure 2: Диаграмма последовательности

Исходный код программы:

main.cpp

```
#include <iostream>
#include <vector>
#include <memory>
#include <fstream>
#include <openssl/evp.h>
#include <iomanip>
#include <sstream>

class Article {
public:
    std::wstring title;
    std::wstring authors;
    std::wstring text;
    std::wstring hash;
};

class ArticleBuilder {
public:
    virtual void reset() = 0;

    virtual void buildTitle(const std::wstring &title) = 0;

    virtual void buildAuthors(const std::vector<std::wstring> &authors) = 0;

    virtual void buildText(const std::wstring &text) = 0;

    virtual void buildHash(const std::wstring &hash) = 0;

    virtual Article getResult() = 0;

    virtual ~ArticleBuilder() = default;
};

class ArticleDirector {
private:
    ArticleBuilder *builder;

public:
    explicit ArticleDirector(ArticleBuilder *builder) : builder(builder) {};

    void construct(
            const std::wstring &title,
            const std::vector<std::wstring> &authors,
            const std::wstring &text,
            const std::wstring &hash
    ) {
        builder->buildTitle(title);
        builder->buildAuthors(authors);
        builder->buildText(text);
        builder->buildHash(hash);
    }
};

class XmlArticleBuilder : public ArticleBuilder {
private:
    Article article;

public:
    XmlArticleBuilder() : article() {}

    void reset() override {
```

```
        article = Article();
    }

    void buildTitle(const std::wstring &title) override {
        article.title = L"  <title>" + title + L"</title>";
    }

    void buildAuthors(const std::vector<std::wstring> &authors) override {
        article.authors = L"  <authors>\n";
        for (int i = 0; i < authors.size(); i++) {
            article.authors += L"    <author>" + authors[i] + L"</author>\n";
        }
        article.authors += L"  </authors>";
    }

    void buildText(const std::wstring &text) override {
        article.text = L"  <text>" + text + L"  </text>";
    }

    void buildHash(const std::wstring &hash) override {
        article.hash = L"  <hash>" + hash + L"</hash>";
    }

    Article getResult() override {
        return article;
    }
};

std::wstring calculateHash(const std::wstring &text) {
    std::string text_utf8(text.begin(), text.end());

    EVP_MD_CTX *mdctx = EVP_MD_CTX_new();
    if (mdctx == nullptr) {
        throw std::runtime_error("Failed to create hash context");
    }

    if (EVP_DigestInit_ex(mdctx, EVP_sha256(), nullptr) != 1) {
        EVP_MD_CTX_free(mdctx);
        throw std::runtime_error("Failed to initialize digest");
    }

    if (EVP_DigestUpdate(mdctx, text_utf8.c_str(), text_utf8.size()) != 1) {
        EVP_MD_CTX_free(mdctx);
        throw std::runtime_error("Failed to update digest");
    }

    unsigned char hash[EVP_MAX_MD_SIZE];
    unsigned int lengthOfHash = 0;
    if (EVP_DigestFinal_ex(mdctx, hash, &lengthOfHash) != 1) {
        EVP_MD_CTX_free(mdctx);
        throw std::runtime_error("Failed to finalize digest");
    }

    EVP_MD_CTX_free(mdctx);

    std::wstringstream wss;
    for (unsigned int i = 0; i < lengthOfHash; ++i) {
        wss << std::setw(2) << std::setfill(L'0') << std::hex << hash[i];
    }

    return wss.str();
}

struct Exception {
    std::wstring message;

```

```
128     explicit Exception(const std::wstring &message) : message(message) {}
129 };
130 
131 void readValues(
132         std::wstring &title,
133         std::vector<std::wstring> &authors,
134         std::wstring &text,
135         std::wstring &hash
136 ) {
137     std::wifstream input(R"(..\..\..\lab3\builder\article.txt)");
138 
139     if (!input.is_open()) {
140         throw Exception(L"Невозможно открыть файл article.txt для чтения."
141 );
142     }
143 
144     getline(input, title);
145 
146     std::wstring all, line;
147     getline(input, line);
148     all += line;
149     line += ' ';
150     for (int l = 0, r = 0; r < line.size(); r++) {
151         if (line[r] == ' ' && l != r) {
152             authors.push_back(line.substr(l, r - l));
153             l = r + 1;
154         }
155     }
156 
157     while (getline(input, line)) {
158         if (input.eof()) {
159             hash = line;
160             break;
161         }
162         all += line + L"\n";
163         text += line + L"\n";
164     }
165 
166     input.close();
167 
168     std::wstring realHash = calculateHash(all);
169     if (hash != realHash) {
170         throw Exception(
171             L"Указанный хэш '" + hash + L"' статьи не совпадает с фактическим '" +
                 realHash + L"'."
172         );
173     }
174 }
175 
176 void writeValues(Article &article) {
177     std::wofstream output(R"(..\..\..\lab3\builder\article.xml)");
178 
179     if (!output.is_open()) {
180         throw Exception(L"Невозможно открыть файл article.xml для записи."
181 );
182     }
183 
184     output << "<article>\n" << article.title << '\n' << article.authors << '\n'
        << article.text << '\n' << article.hash
185             << "\n</article>";
186 
187     output.close();
188 }
189 
190 int main() {
191     setlocale(LC_ALL, "Russian");
```

```
192
193     std::wstring title;
194     std::vector<std::wstring> authors;
195     std::wstring text;
196     std::wstring hash;
197
198     try {
199         readValues(title, authors, text, hash);
200     } catch (Exception &e) {
201         std::wcout << e.message << std::endl;
202         return 0;
203     }
204
205     std::unique_ptr<ArticleBuilder> builder =
            std::make_unique<XmlArticleBuilder>();
206     std::unique_ptr<ArticleDirector> director =
            std::make_unique<ArticleDirector>(builder.get());
207     director->construct(title, authors, text, hash);
208
209     Article article = builder->getResult();
210     builder->reset();
211
212     try {
213         writeValues(article);
214     } catch (Exception &e) {
215         std::wcout << e.message << std::endl;
216     }
217
218     return 0;
219 }
```