Технологии конструирования программного обеспечения

Отчет по лабораторной работе № 3

Группа: 221-3210 **Студент:** Обухов Алексей Сергеевич

Задание на лабораторную работу

1. 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.

Диаграмма классов

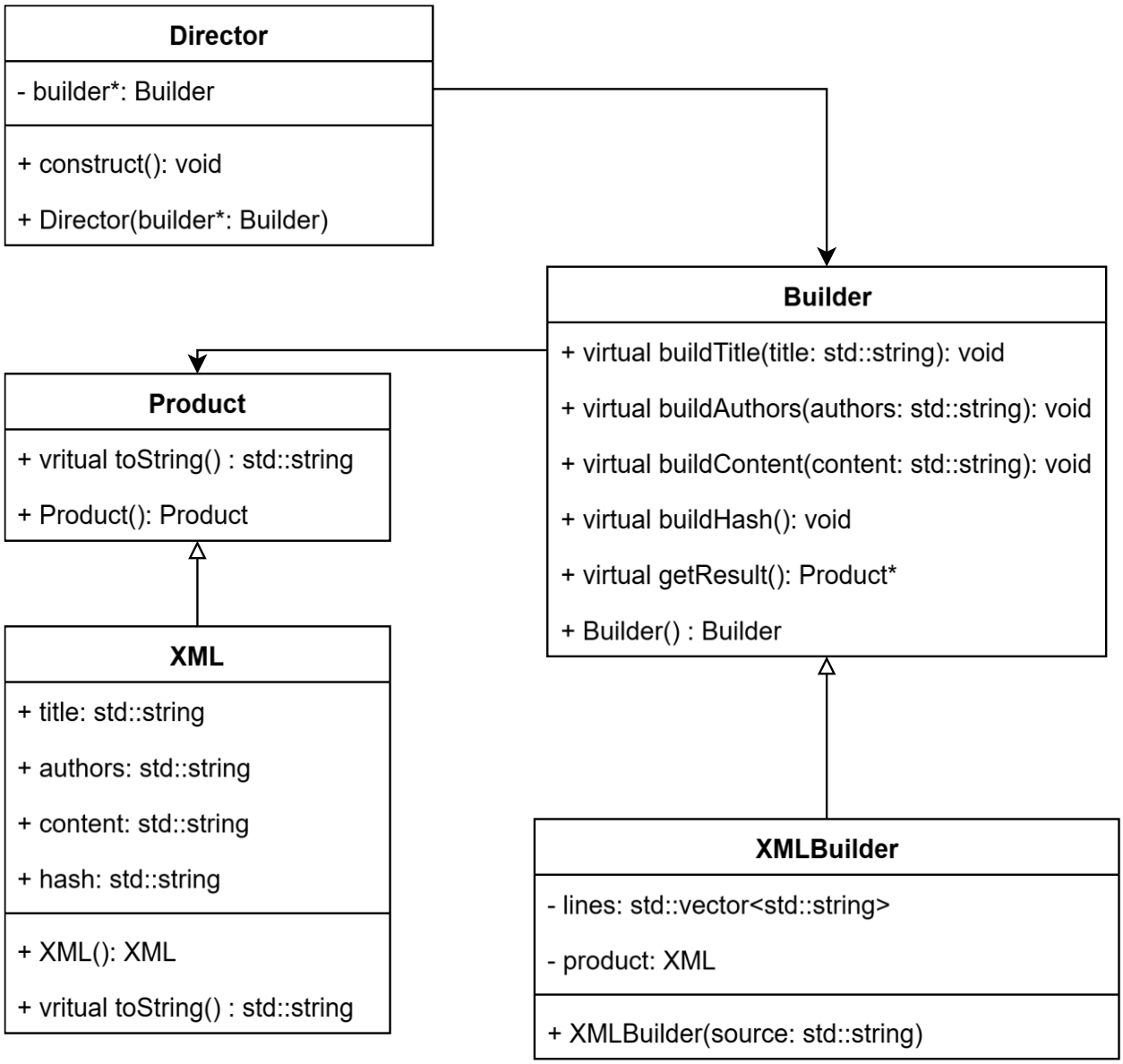
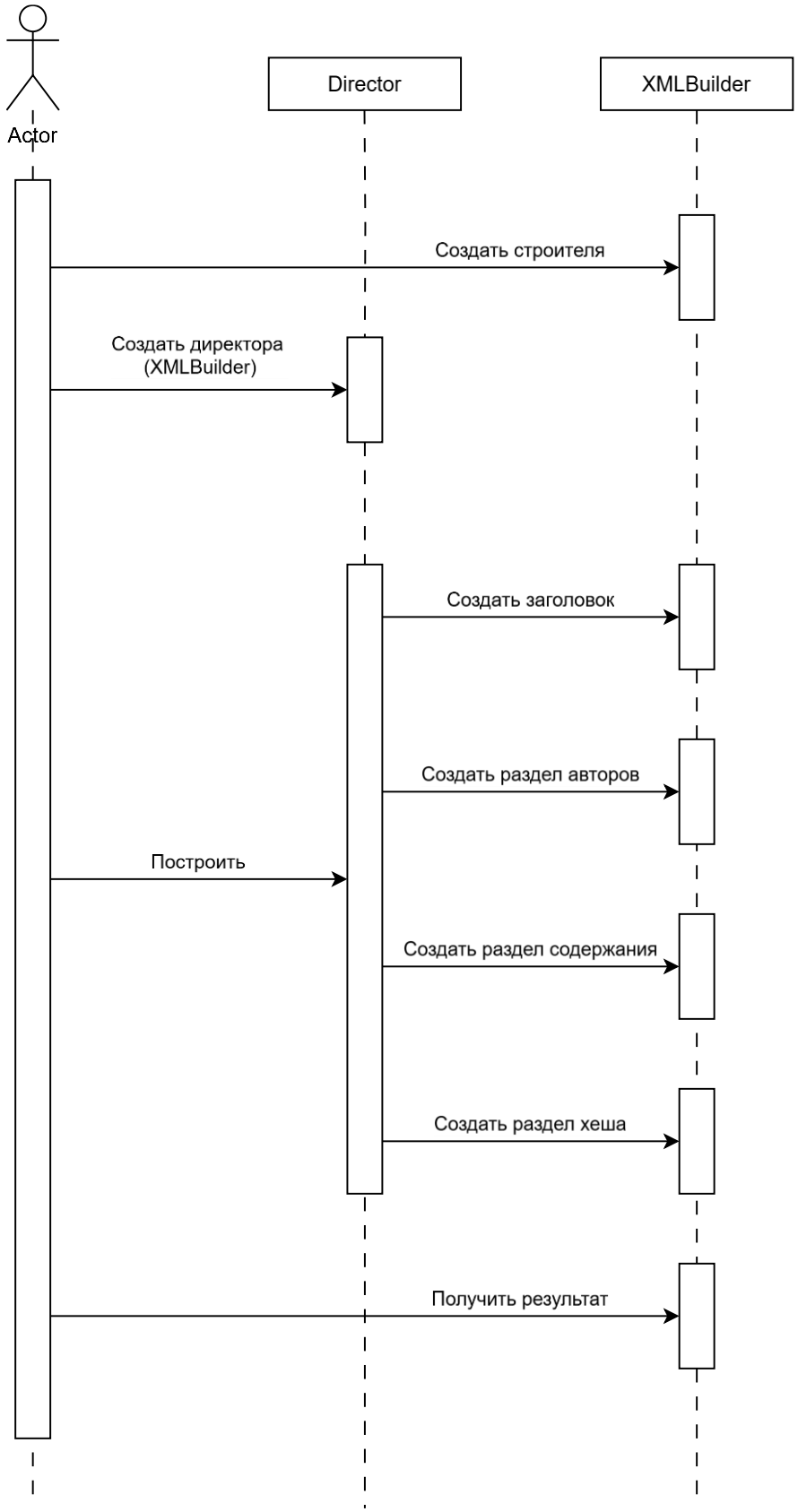


Диаграмма последовательности



Исходный код программы

Файл «Builder.h»

```
#pragma once

#include <string>
#include <vector>
#include <sstream>
#include <cstdlib>

// Базовый класс продукта
class Product {
public:
	Product() {};
	// Методы преобразования продукта в строку
	virtual std::string toString() = 0;
	virtual bool checkHash() = 0;
};

// Класс XML
class XML : public Product {
public:
	std::string title;                // Заголовок
	std::string authors;              // Авторы
	std::string content;              // Содержание
	std::string hash;                 // Хеш-код статьи
	unsigned long long originalHash = 0;   // Хеш, полученный из файла
	unsigned long long calculatedHash = 0; // Хеш, полученный из строителя

	XML() {};

	// Преобразование XML в строку
	std::string toString() override {
		std::string result;
		result += "<?xml version=\"1.0\"?>\n";
		result += "<ARTICLE>\n";
		result += title + "\n";
		result += authors + "\n";
		result += content + "\n";
		result += hash + "\n";
		result += "</ARTICLE>\n";

		return result;
	}

	bool checkHash() override {
		return originalHash == calculatedHash;
	}
};

// Базовый класс строителя
class Builder {
public:
	Builder() {};

	virtual void buildTitle() = 0;
	virtual void buildAuthors() = 0;
	virtual void buildContent() = 0;
	virtual void buildHash() = 0;
	virtual Product* getResult() = 0;
};

// Класс строителя XML
class XMLBuilder : public Builder {
```

```
	// Исходный текст, из которого формируется XML
protected:
	std::vector<std::string> lines;
	XML* product;
public:
	void buildTitle() override {
		product->title = "<TITLE>" + lines[0] + "</TITLE>";
	};

	void buildAuthors() override {
		product->authors = "<AUTHORS>" + lines[1] + "</AUTHORS>";
	};

	void buildContent() override {
		product->content = "<CONTENT>\n";
		for (int i = 2; i < lines.size() - 1; i++)
			product->content += lines[i] + "\n";
		product->content += "</CONTENT>";
	};

	void buildHash() override {
		std::hash<std::string> hasher;
		std::string fileContent;

		for (int i = 0; i < lines.size() - 1; i++)
			fileContent += lines[i];

		unsigned long long calculatedHash = hasher(fileContent);
		product->hash = "<HASH>" + lines[lines.size() - 1] + "</HASH>";
		product->originalHash = std::stoull(lines[lines.size() - 1]);
		product->calculatedHash = calculatedHash;
	};

	Product* getResult() override {
		return product;
	}

	XMLBuilder(std::string source) {
		product = new XML();
		std::stringstream ss(source);
		std::string line;
		while (std::getline(ss, line)) {
			if (!line.empty())
				lines.push_back(line);
		}
	}
};

// Класс директора
class Director {
protected:
	Builder* builder;

public:
	Director(Builder* builder) {
		this->builder = builder;
	}

	void construct() {
		builder->buildTitle();
		builder->buildAuthors();
		builder->buildContent();
		builder->buildHash();
	}
};
```

Файл «main.cpp»

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

#include "Builder.h"

std::string readFile(std::string path) {
	std::string content;

	std::ifstream file(path);

	for (std::string line; std::getline(file, line);)
		content += line + "\n";

	file.close();

	return content;
}

// Здесь можно вычислить хеш статьи
// https://toolkitbay.com/tkb/tool/FNV-1
// Алгоритм: FNV-1 64A
// ВАЖНО! Обязательно загружать файл в кодировке ANSI,
//             Предварительно удалив ВСЕ переносы строк

int main() {
	setlocale(LC_ALL, "Russian");

	std::string fileContent = readFile("test.txt");
	std::cout << "Оригинал файла:\n";
	std::cout << fileContent << "\n";

	XMLBuilder builder(fileContent);
	Director director(&builder);
	director.construct();
	Product* xml = builder.getResult();

	std::cout << "Полученный XML:\n";
	std::cout << xml->toString() << "\n";
	std::cout << "XML хеш верный?: " << (xml->checkHash() ? "Да" : "Нет") <<
"\n";

	return 0;
}
```

2. 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.

Диаграмма классов

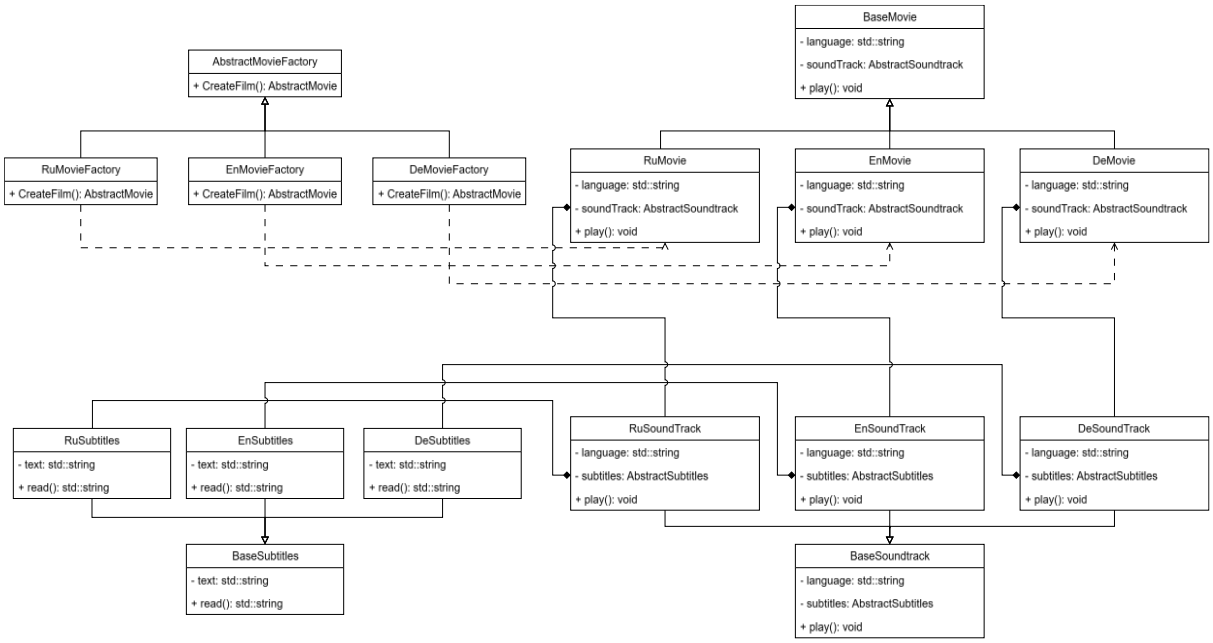
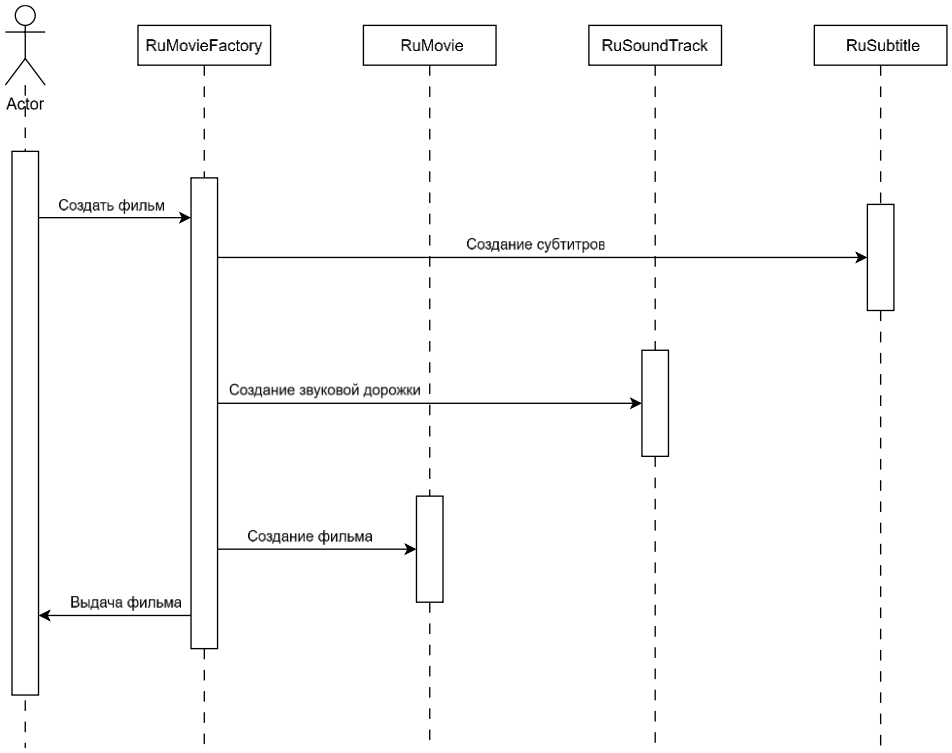


Диаграмма последовательности



Исходный код программы

Содержимое «Factory.h»

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

// Базовый класс для субтитров
class BaseSubtitle {
protected:
	std::string text;

public:
	std::string read() {
		return text;
	}

	BaseSubtitle() {
		text = "Базовый текст базовых субтитров";
	}
};

// Субтитры на русском языке
class RuSubtitle : public BaseSubtitle {
public:
	RuSubtitle() {
		text = "Субтитры на русском языке";
	}
};

// Субтитры на английском языке
class EnSubtitle : public BaseSubtitle {
```

```
public:
	EnSubtitle() {
		this->text = "Subtitles in English";
	}
};

// Субтитры на немецком языке
class DeSubtitle : public BaseSubtitle {
public:
	DeSubtitle() {
		this->text = "Untertitel auf Deutsch";
	}
};


// Базовый класс звуковой дорожки
class BaseSoundtrack {
protected:
	std::string language;
	BaseSubtitle* subtitle;

public:
	BaseSoundtrack() {
		this->language = "NoLang";
		this->subtitle = nullptr;
	}

	// Воспроизведение звуковой дорожки
	void play() {
		std::cout << "Язык звуковой дорожки: " << language << "\n";
		std::cout << "Текст субтитров:\n";
		std::cout << subtitle->read() << "\n";
	}
};


// Звуковая дорожка на русском языке
class RuSoundtrack : public BaseSoundtrack {
public:
	RuSoundtrack(RuSubtitle* subtitle) {
		this->language = "Русский";
		this->subtitle = subtitle;
	};
};

// Звуковая дорожка на английском языке
class EnSoundtrack : public BaseSoundtrack {
public:
	EnSoundtrack(EnSubtitle* subtitle) {
		this->language = "English";
		this->subtitle = subtitle;
	};
};

// Звуковая дорожка на английском языке
class DeSoundtrack : public BaseSoundtrack {
public:
	DeSoundtrack(DeSubtitle* subtitle) {
		this->language = "Deutsch";
		this->subtitle = subtitle;
	};
};

// Базовый класс фильма
class BaseMovie {
```

```
protected:
	std::string language;
	BaseSoundtrack* soundtrack;

public:
	BaseMovie() {
		language = "NoLang";
		soundtrack = nullptr;
	}

	void play() {
		std::cout << "Язык фильма: " << language << "\n";
		std::cout << "Звуковая дорожка:\n";
		soundtrack->play();
	}
};

// Класс фильма на русском языке
class RuMovie : public BaseMovie {
public:
	RuMovie(RuSoundtrack* soundtrack) {
		this->language = "Русский";
		this->soundtrack = soundtrack;
	}
};

// Класс фильма на английском языке
class EnMovie : public BaseMovie {
public:
	EnMovie(EnSoundtrack* soundtrack) {
		this->language = "English";
		this->soundtrack = soundtrack;
	}
};

// Класс фильма на немецокм языке
class DeMovie : public BaseMovie {
public:
	DeMovie(DeSoundtrack* soundtrack) {
		this->language = "Deutsch";
		this->soundtrack = soundtrack;
	}
};


// Абстрактная фабрика фильмов
class AbstractMovieFactory {
public:
	virtual BaseMovie* createMovie() = 0;
};

// Фабрика фильмов на русском языке
class RuMovieFactory : public AbstractMovieFactory {
public:
	BaseMovie* createMovie() {
		RuSubtitle* subtitle = new RuSubtitle();
		RuSoundtrack* soundtrack = new RuSoundtrack(subtitle);
		RuMovie* movie = new RuMovie(soundtrack);
		return movie;
	}
};

// Фабрика фильмов на английском
class EnMovieFactory : public AbstractMovieFactory {
public:
```

```
	BaseMovie* createMovie() {
		EnSubtitle* subtitle = new EnSubtitle;
		EnSoundtrack* soundtrack = new EnSoundtrack(subtitle);
		EnMovie* movie = new EnMovie(soundtrack);
		return movie;
	}
};
// Фабрика фильмов на немецком
class DeMovieFactory : public AbstractMovieFactory {
public:
	BaseMovie* createMovie() {
		DeSubtitle* subtitle = new DeSubtitle;
		DeSoundtrack* soundtrack = new DeSoundtrack(subtitle);
		DeMovie* movie = new DeMovie(soundtrack);
		return movie;
	}
};
```

Содержимое «main.cpp»

```
#include <iostream>
#include "Factory.h"

int main() {
	setlocale(LC_ALL, "Russian");

	RuMovieFactory ruMovieFactory;
	EnMovieFactory enMovieFactory;
	DeMovieFactory deMovieFactory;

	BaseMovie* movie;

	// Создаем и проигрываем фильм на русском
	movie = ruMovieFactory.createMovie();
	movie->play();
	std::cout << "\n\n";

	// Создаем и проигрываем фильм на английском
	movie = enMovieFactory.createMovie();
	movie->play();
	std::cout << "\n\n";

	// Создаем и проигрываем фильм на немецком
	movie = deMovieFactory.createMovie();
	movie->play();

	return 0;
}
```

3. 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.

Диаграмма классов

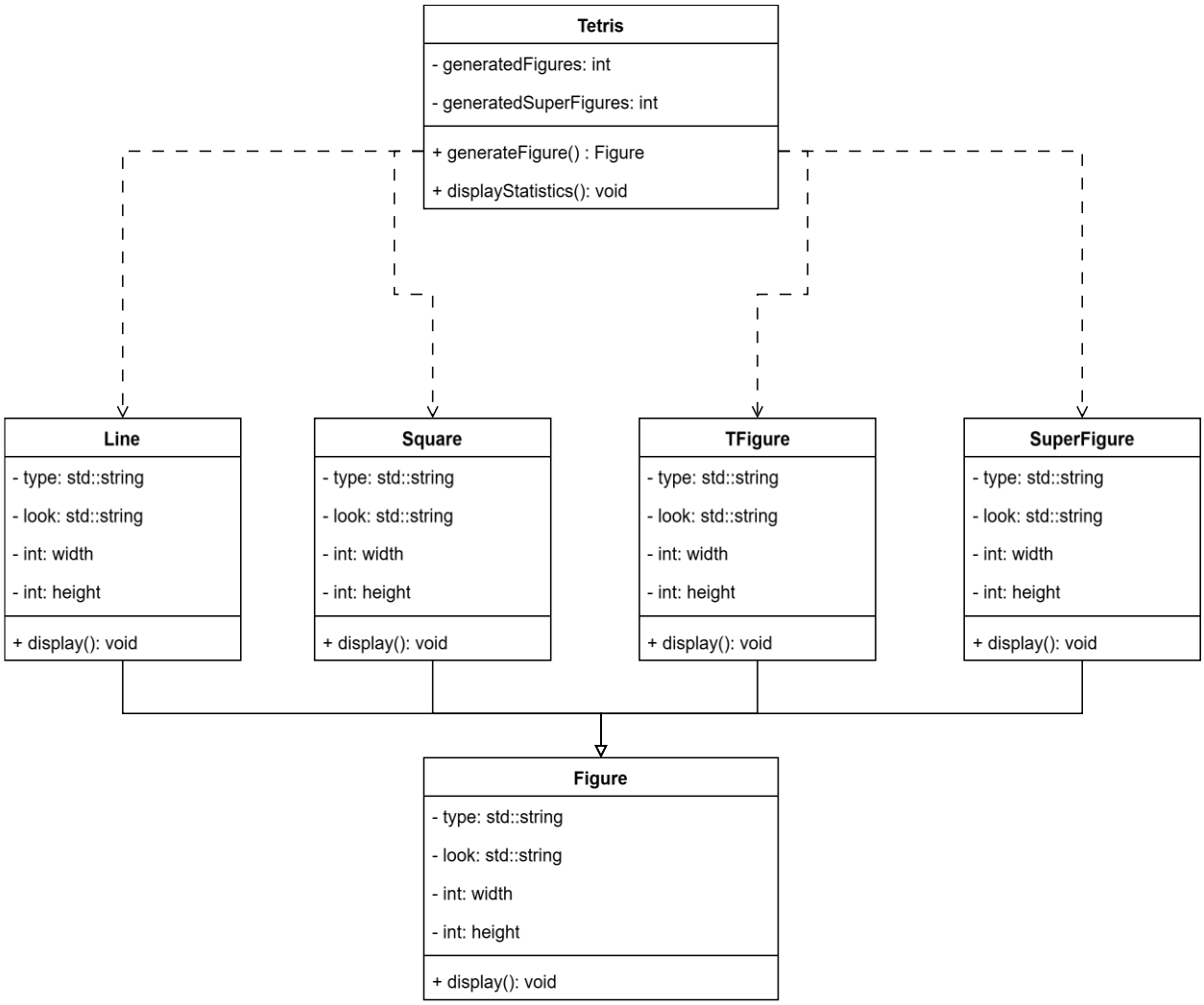
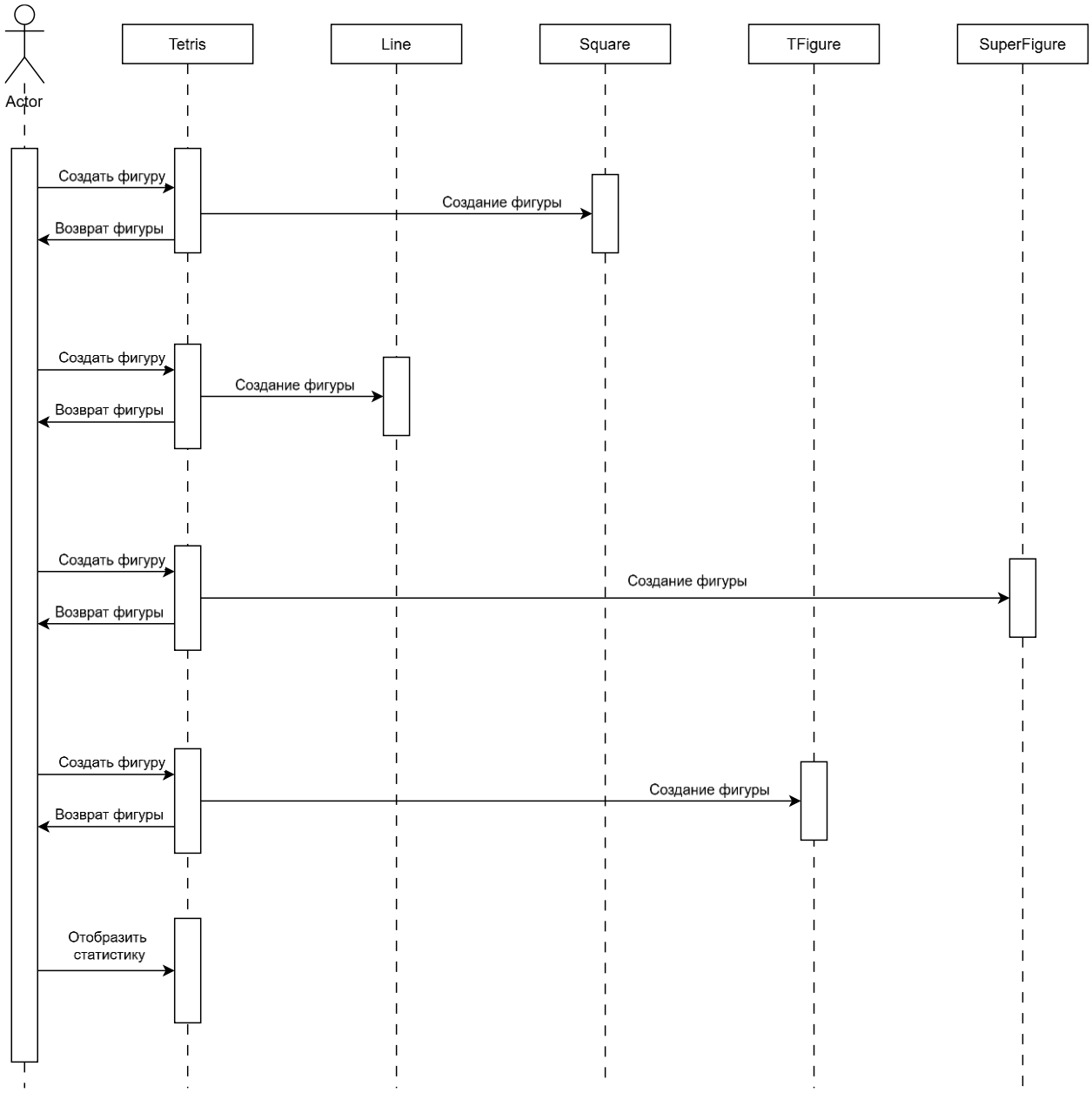


Диаграмма последовательности



Исходный код программы

Содержимое «FactoryMethod.h»

```
#pragma once

#include <iostream>
#include <string>


// Базовый класс фигуры
class Figure {
	friend class Tetris;
protected:
	std::string type;
	std::string look;
	int width = 0, height = 0;
```

```
public:
	Figure() {};
	void display() {
		std::cout << this->type << " : " << this->look << " (" << this->height
<< ", " << this->width << ")\n";
	}
};

//Класс фигуры "палка"
class Line : public Figure {
	friend class Tetris;
	Line() {
		this->width = 1;
		this->height = 4;
		this->look = "|";
		this->type = "line";
	}
};

// Класс фигуры "квадрат 2 на 2"
class Square : public Figure {
	friend class Tetris;
	Square() {
		this->width = 2;
		this->height = 2;
		this->look = "[]";
		this->type = "square";
	}
};

// Класс Т-образной фигуры
class TFigure : public Figure {
	friend class Tetris;
	TFigure() {
		this->width = 3;
		this->height = 2;
		this->look = "т";
		this->type = "t_figure";
	}
};


// Класс большой фигуры
class SuperFigure : public Figure {
	friend class Tetris;
	SuperFigure() {
		this->width = 5;
		this->height = 5;
		this->look = "-|||-";
		this->type = "super";
	}
};

// Класс игры тетрис
class Tetris {
private:
	unsigned int generatedFigures = 0;
	unsigned int generatedSuperFigures = 0;


public:
	Tetris() {
		srand(time(NULL));
	}
```

```
	// Функция генерации фигур
	Figure generateFigure() {
		int random = rand() % 4;

		this->generatedFigures++;

		switch (random)
		{
		case 0:
			return Line();
			break;
		case 1:
			return Square();
			break;
		case 2:
			return TFigure();
			break;
		case 3:
			this->generatedSuperFigures++;
			return SuperFigure();
			break;
		default:
			break;
		}
	}

	// Вывод статистики
	void displayStatistics() {
		std::cout << "Статистика:\n";
		std::cout << "Создано фигур: " << this->generatedFigures << "\n";
		std::cout << "Из них суперфигур: " << this->generatedSuperFigures <<
"\n";
	}
};
```

Содержимое «main.cpp»

```
#include <iostream>
#include "FactoryMethod.h"

int main() {
	setlocale(LC_ALL, "Russian");

	Tetris tetris;
	Figure figure;

	for (int i = 0; i < 5; i++) {
		figure = tetris.generateFigure();
		figure.display();
	}

	std::cout << "\n";

	tetris.displayStatistics();

	return 0;
}
```