Технологии конструирования программного обеспечения

Отчет по лабораторной работе № 02

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

Задание на лабораторную работу:

Для заданного варианта задания (предметной области) разработать UML-диаграмму классов и диаграмму последовательности. Разработать программу решения задания в виде консольного приложения (C++, C#) с использованием принципа подстановки Лисков (LSP), принципа разделения интерфейсов (ISP) и принципа инверсии (DIP). Допускается вводить дополнительные понятия предметной области. Наделите классы атрибутами и функциональностью по вашему усмотрению. В программе предусмотрите тестирование функциональности созданных объектов классов.

Номер в списке - 15, вариант 2:

Организация, страховая компания, нефтегазовая компания, завод.

UML-диаграмма классов:

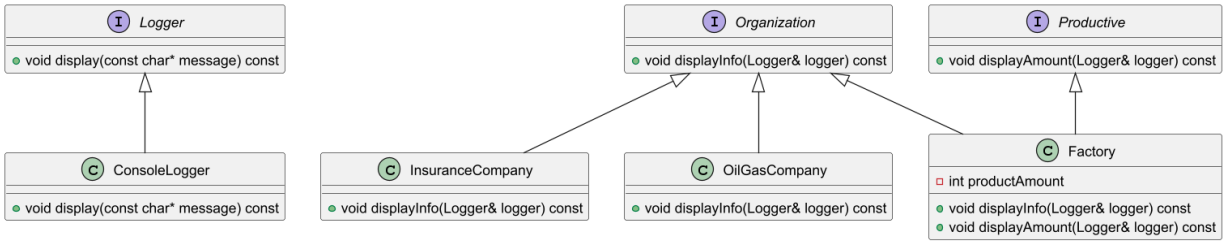

Figure 1: UML-диаграмма классов

Диаграмма последовательности:

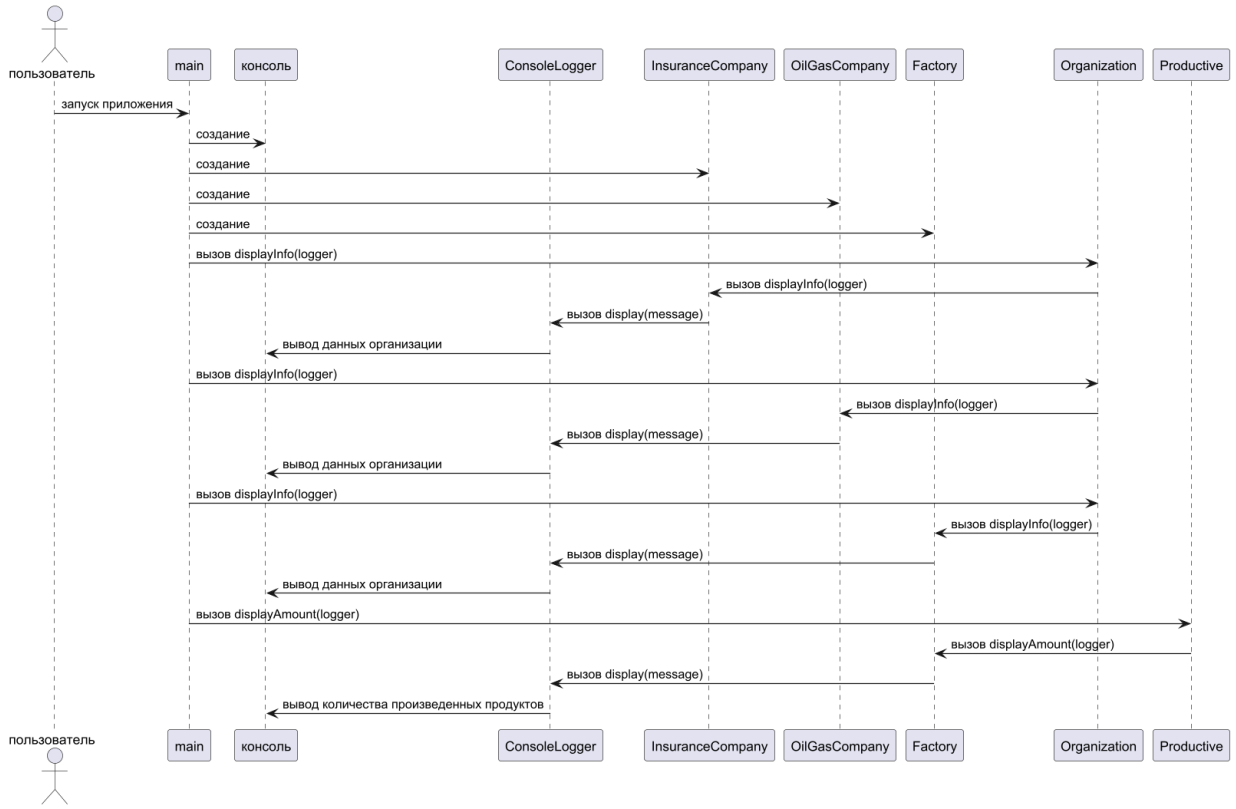

Figure 2: Диаграмма последовательности

Исходный код программы:

main.cpp

```cpp
#include <iostream>
#include <vector>
#include <memory>

using std::cout, std::endl, std::make_unique, std::unique_ptr, std::vector;

class Logger {
public:
    virtual ~Logger() = default;

    virtual void display(const char *message) const = 0;
};

class ConsoleLogger : public Logger {
public:
    void display(const char *message) const override {
        std::cout << message << std::endl;
    }
};

class Organization {
public:
    virtual ~Organization() = default;

    virtual void displayInfo(Logger &logger) const = 0;
};

```

```cpp
// принцип разделения интерфейсов
class Productive {
public:
    virtual ~Productive() = default;

    virtual void displayAmount(Logger &logger) const = 0;
};

class InsuranceCompany : public Organization {
public:
    void displayInfo(Logger &logger) const override {
        // принцип инверсии зависимостей
        logger.display("This is an insurance company.");
    }
};

class OilGasCompany : public Organization {
public:
    void displayInfo(Logger &logger) const override {
        logger.display("This is an oil and gas company.");
    }
};

class Factory : public Organization, public Productive {
private:
    int productAmount;
public:
    explicit Factory(int amount) : productAmount(amount) {}

    void displayInfo(Logger &logger) const override {
        logger.display("This is a factory.");
    }

    void displayAmount(Logger &logger) const override {
        logger.display(("Product amount: " + std::to_string(productAmount) +
            ".").c_str());
    }
};

int main() {
    ConsoleLogger logger;

    // принцип подстановки Лисков
    vector<unique_ptr<Organization>> organizations;

    organizations.push_back(make_unique<InsuranceCompany>());
    organizations.push_back(make_unique<OilGasCompany>());
    organizations.push_back(make_unique<Factory>(100));

    for (const auto &organization: organizations) {
        organization->displayInfo(logger);
    }

    unique_ptr<Productive> productive = make_unique<Factory>(200);
    productive->displayAmount(logger);

    return 0;
}
```