МИНИСТЕРСТВО ОБРАЗОВАНИЯ, КУЛЬТУРЫ И ИССЛЕДОВАНИЯ РЕСПУБЛИКИ МОЛДОВА

ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ МОЛДОВЫ

ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ, ИНФОРМАТИКИ И МИКРОЭЛЕКТРОНИКИ

ДЕПАРТАМЕНТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И АВТОМАТИКИ

ОТЧЕТ

По лабораторной работе №2

по дисциплине:

Объектно-ориентированный анализ, проектирование и программирование

Тема: Принципы SOLID. Single Responsibility и Interface Segregation

Выполнил: студент группы TI-154 Лукаш Сергей

Проверил: Пекарь Михаил

**Цель работы**:

Изучение принципов SOLID. Single Responsibility и Interface Segregation

**Ход работы**

Исходный код проекта можно найти по следующему адресу:

* «плохой» вариант: <https://github.com/SergeyLucash/APPOO/tree/master/Lab2>
* исправленный вариант: <https://github.com/SergeyLucash/APPOO/tree/lab2_fix/Lab2>

В процессе изучения были созданы два варианта проекта. Плохой пример отражает нарушение двух принципов SOLID: Single Responsibility и Interface Segregation.

public interface IRepository

{

void InitializeData();

IEnumerable<Movie> GetMoviesFromFile();

IEnumerable<Movie> GetMoviesFromList();

Movie GetMovieByIdFromFile(int id);

Movie GetMovieByIdFromList(int id);

void Log(string message);

void UpdateInFile(Movie movie);

void UpdateInList(Movie movie);

void DeleteFromFile(int id);

void DeleteFromList(int id);

}

Данный интерфейс нарушает два принципа. Single Responsibility нарушается тем, что классы, реализующие его реализуют поведение, не связанное с основной целью класса, так как на него возложены обязанности по работе с файлами, списками, а также логами. Interface Segregation нарушается тем, что данный интерфейс обладает большим количеством методов. Так, классам, которые должны работать только с файлами, должны будут также реализовывать и работу логгера, и работу со списком.

Решение данной проблемы следующее: разделение на три интерфейса IRepository – отвечающий за работу с данными, IDataProvider – источник данных, и ILogger, а также создание двух классов источника данных – для файла и для списка.

public interface IRepository<T>

{

IEnumerable<T> GetAllMovies();

T GetById(int id);

void Add(T entinty);

void Update(T entity);

void Delete(int id);

}

public interface ILogger

{

void Log(string message);

}

public interface IDataProvider<T>

{

IList<T> GetData();

void SaveChanges();

}

**Результат работы**

Приведем скриншоты работы программы для шифра Цезаря: выбор метода (рис. 1), шифрование (рис. 2), дешифровка (рис. 3).

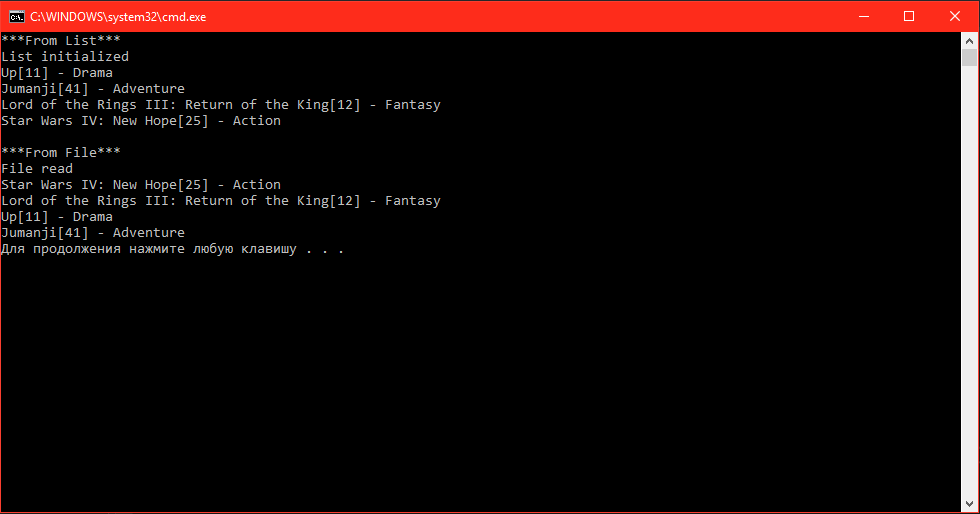


Рисунок 1 – Результаты работы «плохого» варианта

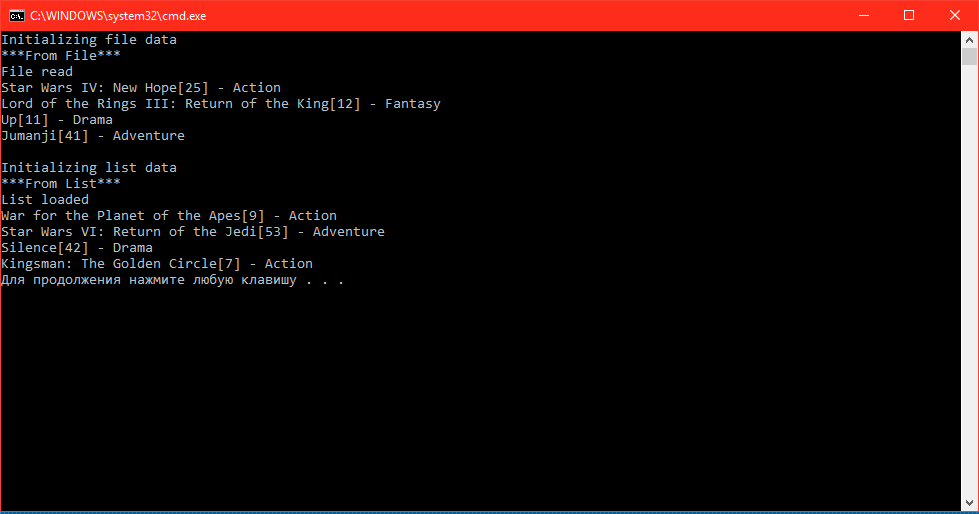


Рисунок 2 – Результаты работы исправленного варианта

**Вывод**

При выполнении данной лабораторной работы мы изучили некоторые принципы SOLID, а именно: принцип единственной ответственности и принцип разделения интерфейса.

Для примера использования всех данных принципов на практике, был создан проект, представляющий собой простейший репозиторий фильмов. Были написаны два варианта проекта – «плохой», который нарушает принципы, и исправленный – где принципы не нарушены. Хотя результат работы обоих вариантов одинаков, исправленный обладает большей гибкостью и расширяемостью. Так, при необходимости создания нового типа репозитория, или в случае необходимости только одного типа репозитория, к примеру, основанного на базе данных, необходимо будет реализовать только один интерфейс (IRepository), без необходимости реализовывать функционал логгера, работы с файлами и списками. То же касается и логгера. Для создания нового логгера нет необходимости реализовывать ненужный функционал, все что необходимо – реализовать интерфейс ILogger.