ГУАП

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| канд. техн. наук, доцент |  |  |  | Н. В. Богословская |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 6 |
| МЕХАНИЗМЫ НАСЛЕДОВАНИЯ: КОНСТРУКТОРЫ ПРОИЗВОДНЫХ КЛАССОВ |
| по курсу: |
| ОСНОВЫ ПРОГРАММИРОВАНИЯ |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4326 |  |  |  | Г. С. Томчук |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc185591438)

[2 Задача 3](#_Toc185591439)

[3 Ключевые позиции 3](#_Toc185591440)

[3.1 Реализация методов 3](#_Toc185591441)

[3.2 Создание интерфейса 5](#_Toc185591442)

[4 Тестирование программы 5](#_Toc185591443)

[ВЫВОДЫ 8](#_Toc185591444)

1. Цель работы

Выполнение работы имело следующие цели: изучить механизмы наследования в объектно-ориентированном программировании, включая реализацию конструкторов производных классов, работу с восходящими и нисходящими преобразованиями типов, а также особенности наследования от типа Object.

1. Задача

11. В приложении пользователь может создать объект класса Текстовый файл, используя классы Файл, Директория. Методы для работы с файлами должны полностью обеспечить пользователя возможностями создания, удаления, изменения, переименования файлов. Подзадачи лабораторной работы № 6 включают в себя:

* Добавить и протестировать необходимые конструкторы в классах-наследниках, с учетом особенностей базового класса.
* Реализовать новые поля, свойства и методы для созданных классов, подтвердив, что все они неявно наследуют от типа Object.
* Выполнить восходящее и нисходящее преобразования типов, показав:
  + Ограничения базового класса в доступе к полям производного класса.
  + Корректность преобразований объектов различных типов.

1. Ключевые позиции
   1. Реализация методов

На рис. 1, 2 показаны производные от BrowserFileSystemItem классы BrowserFile, BrowserDirectory, их конструкторы и т.д.

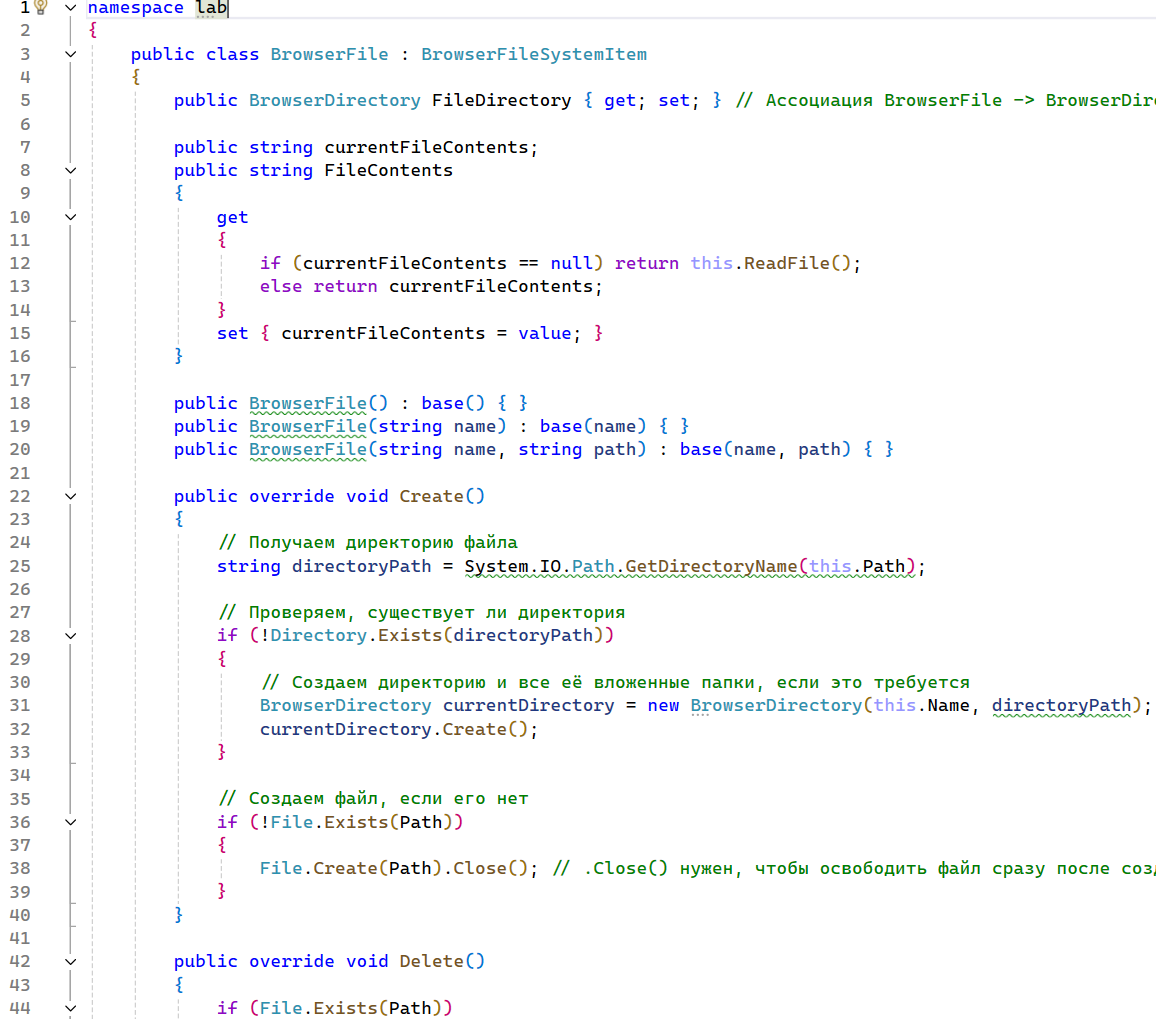


Рисунок 1

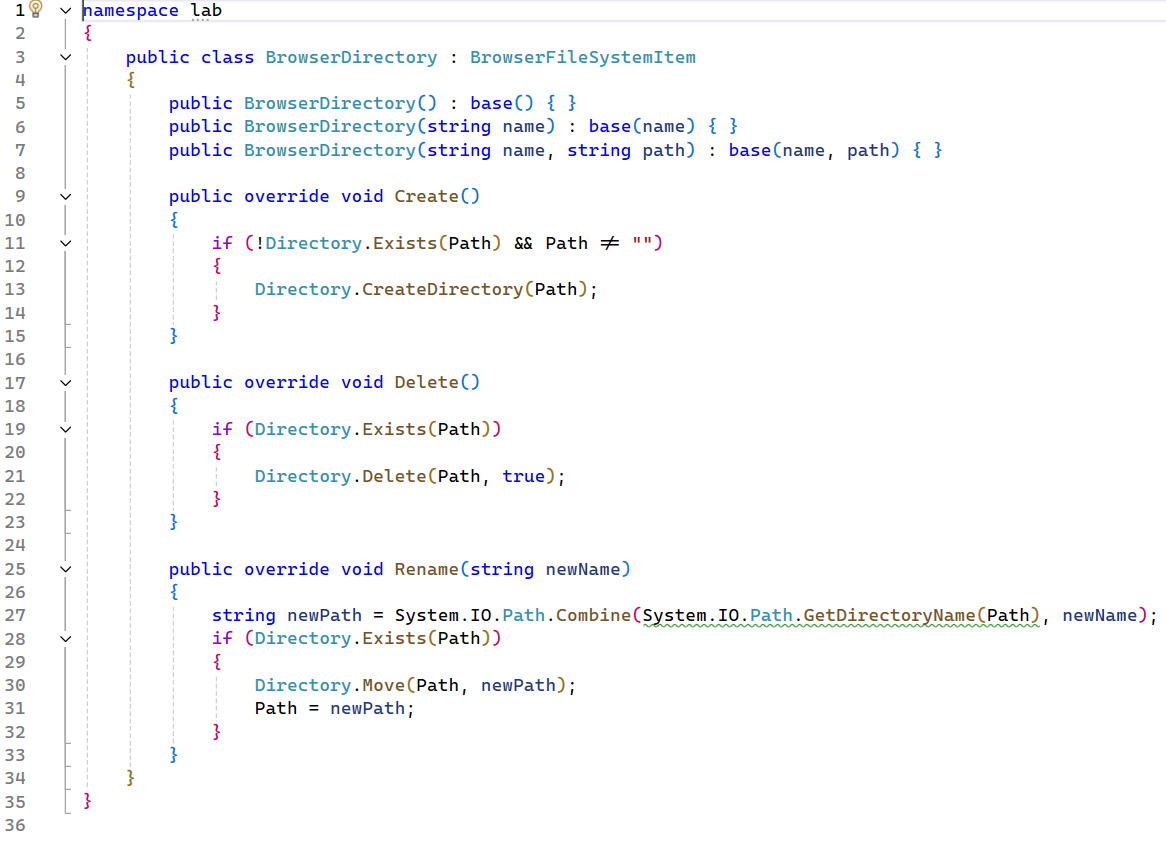


Рисунок 2

На рис. 3 изображена диаграмма классов библиотеки.

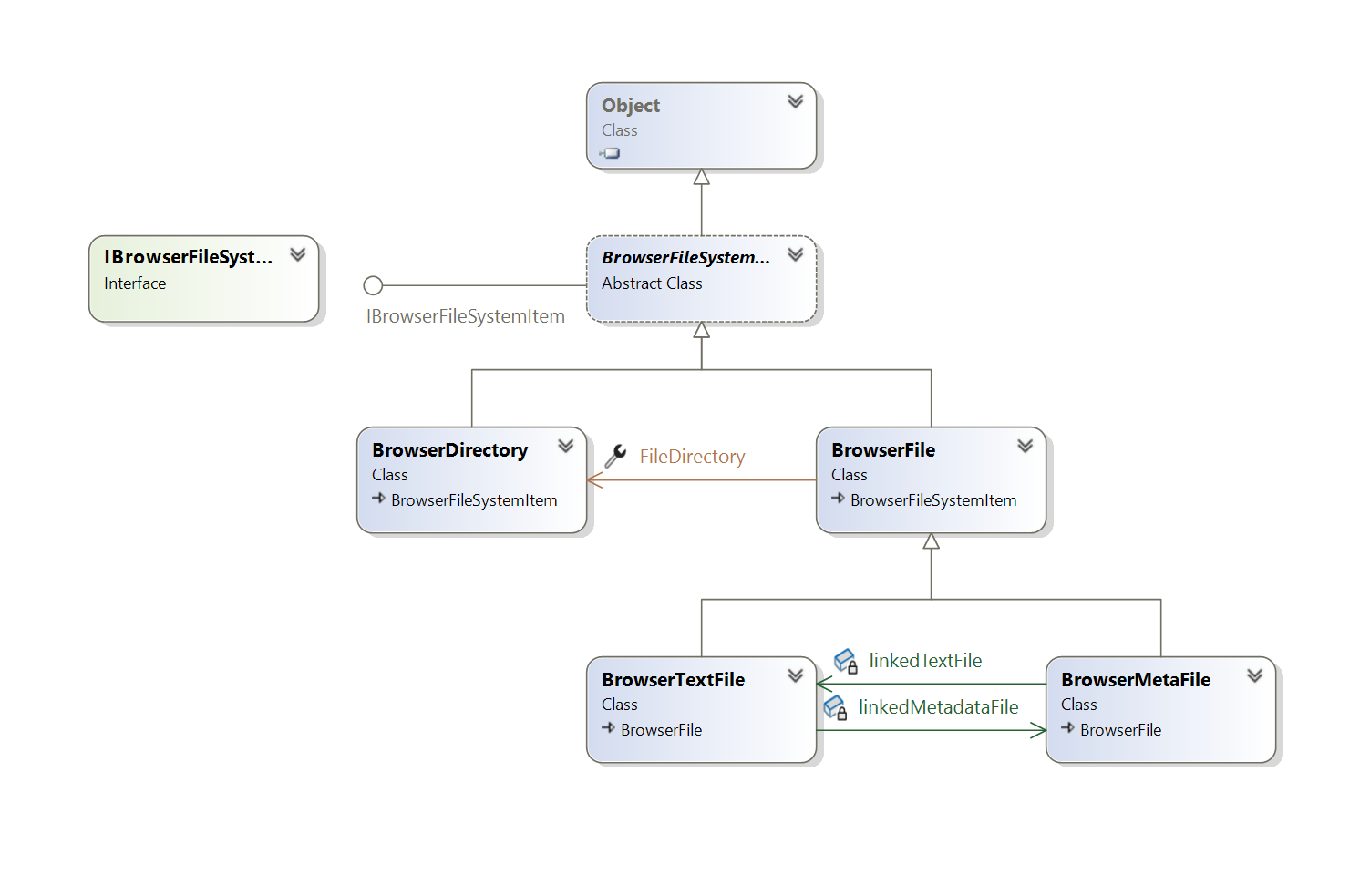


Рисунок 3

* 1. Создание интерфейса

На рис. 4 показан метод для тестирования объектов производных классов и их UpCasting, DownCasting преобразований.

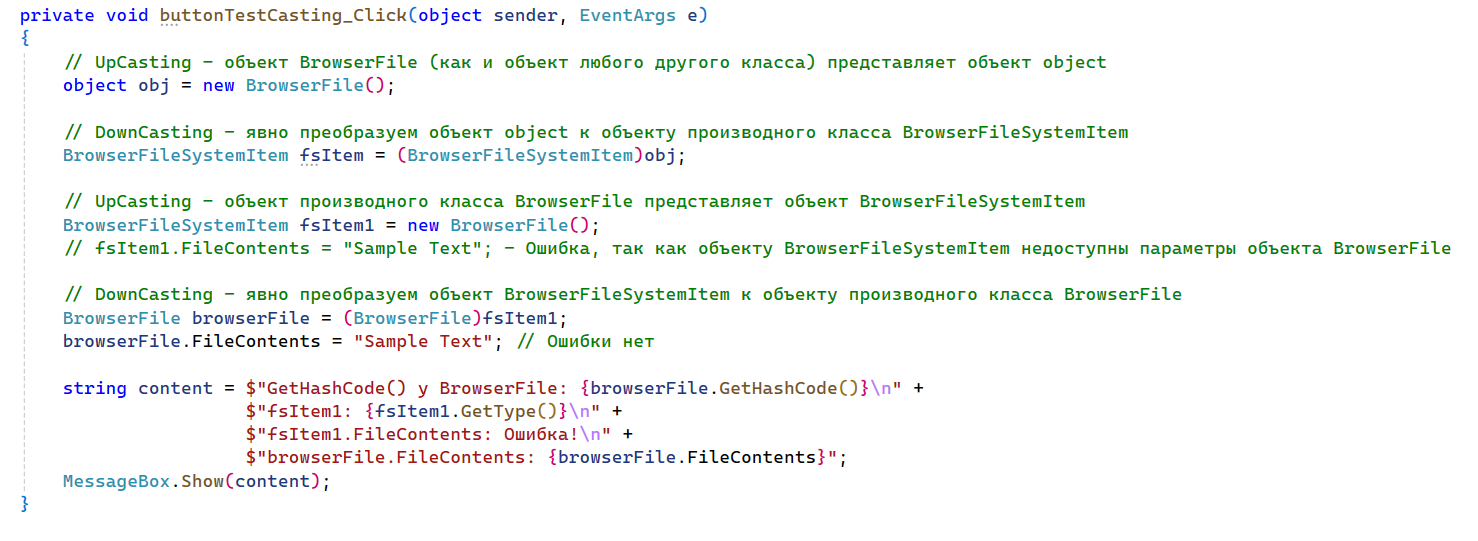


Рисунок 4

1. Тестирование программы

На рис. 5, 6 представлены результаты тестирования программы. На рис. 5 изображено основное окно программы.

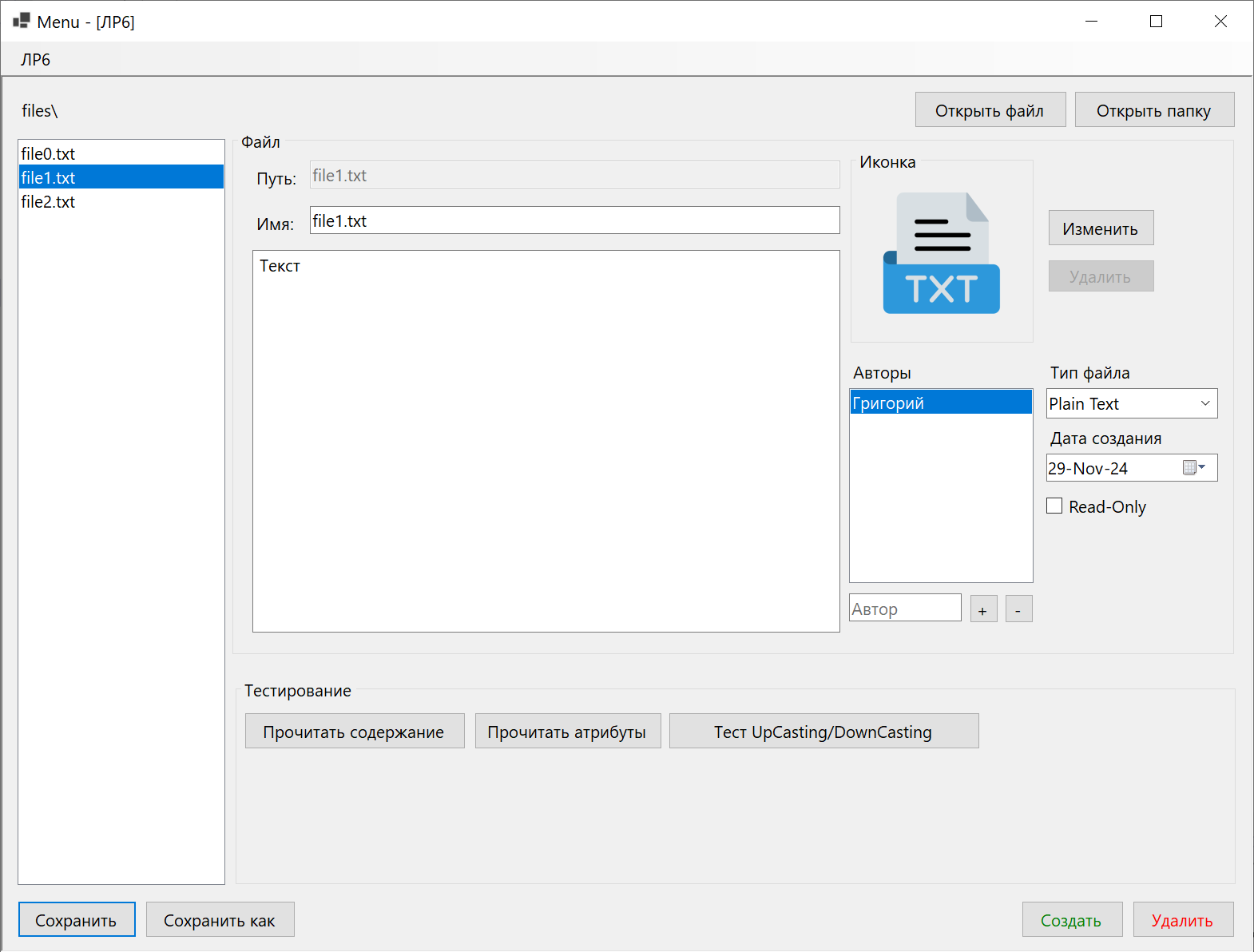


Рисунок 5

На рис. 6 показано окно вызова метода тестирования объектов производных классов и их UpCasting, DownCasting преобразований. Выведенная информация говорит о том, что BrowserFile, как и все остальные классы неявно происходят от класса Object. UpCasting - объект производного класса BrowserFile представляет объект fsItem1 класса BrowserFileSystemItem, однако объекту fsItem1 недоступны параметры объекта browserFile. DownCasting - явно преобразуем объект fsItem1 к объекту производного класса BrowserFile и тем самым получаем доступ к параметрам объекта browserFile.

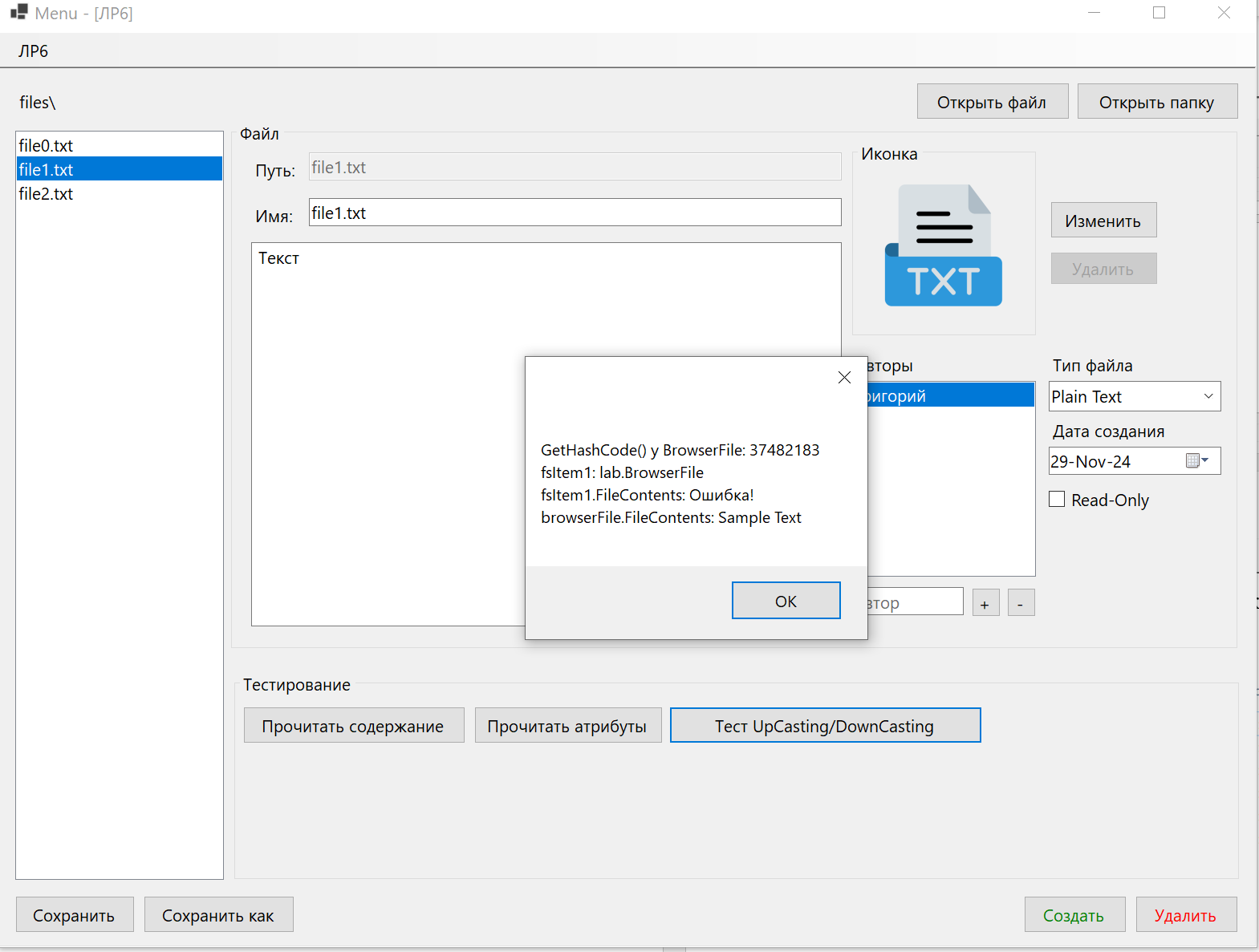


Рисунок 6

ВЫВОДЫ

* Конструкторы в наследуемых классах: Конструкторы производных классов могут использовать базовые конструкторы через ключевое слово base, что позволяет избегать дублирования кода и обеспечивает согласованность между классами.
* Наследование от Object: Все классы в C# неявно наследуются от Object, что подтверждается наличием у любого класса методов ToString(), GetHashCode() и других, доступных по умолчанию.
* Преобразования типов:
  + Восходящее преобразование (из производного класса в базовый) безопасно и не требует явного указания. Однако базовый класс теряет доступ к уникальным полям и методам производного класса.
  + Нисходящее преобразование (из базового класса в производный) требует явного указания и проверки, что объект действительно является экземпляром производного класса. В противном случае возникает исключение.
* Практическая значимость: Использование наследования и преобразований типов позволяет эффективно проектировать системы, где объекты разных типов могут обрабатываться единообразно, сохраняя при этом гибкость и расширяемость кода.