Оглавление

Цели и задачи лабораторной работы

Целью работы является приобретение навыков разработки структуры программных средств в соответствии с выбранным архитектурным шаблоном с использованием канонической диаграммы классов UML.

Решаемая в ходе выполнения работы задача – построение диаграммы классов проектируемого программного продукта согласно варианту №7 «Графический текстовый редактор».

1. ОСНОВНЫЕ ТРЕБОВАНИЯ К РАЗРАБАТЫВАЕМОМУ ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ

В данном случае в качестве искомого программного продукта будет рассматриваться простейший вариант графического текстового редактора, представляющий собой оконное приложение.

К основным функциональным требованиям можно отнести следующие:

1) у пользователя должна быть возможность открытия и сохранения текстовых файлов в формате *.txt;

2) у пользователя должна быть возможность редактирования (создания, изменения, удаления) текста;

3) у пользователя должна быть возможность поиска и замены символов, слов и выражений.

К основным нефункциональным требованиям можно отнести следующие:

1) приложение должно быть способно выполняться как в операционных системах Windows, так и в Linux без изменений;

2) интерфейс системы должен быть интуитивно понятным и доступным для пользователей:

– дублирование основных функций меню с помощью кнопок на панели инструментов основного окна;

– дублирование основных функций меню с помощью комбинаций клавиш («горячих» клавиш).

2. ВЫБОР АРХИТЕКТУРНЫХ СТИЛЕЙ

Архитектура программной системы практически никогда не ограничена лишь одним архитектурным стилем, зачастую она является сочетанием архитектурных стилей, образующих полную систему.

Для реализации перечисленных требований к графическому текстовому редактору наиболее подходят следующие архитектурные стили.

Объектно-ориентированная архитектура (тип модульного представления) – парадигма проектирования, основанная на распределении ответственности приложения или системы между отдельными многократно используемыми и самостоятельными объектами, содержащими данные и поведение.

Шаблон представления с отделением (является как разновидностью многослойного стиля, так и одним из вариантов архитектуры «публикация-подписка») «Модель-Представление-Контроллер» (или «Model-View-Controller», или «MVC»).

Архитектура MVC стремится отделить то, что видит пользователь (представление), от внутреннего состояния (модели) и от совокупности взаимодействий (контроллер), которые вызывают изменения в этих частях.

Паттерн MVC предоставляет мощную, но хорошо управляемую архитектуру, которая подходит для разработки любых десктопных приложений.

3. РАЗРАБОТКА ДИАГРАММЫ КЛАССОВ

В рамках выбранной предметной области была разработана следующая диаграмма классов (рис. 3.1).

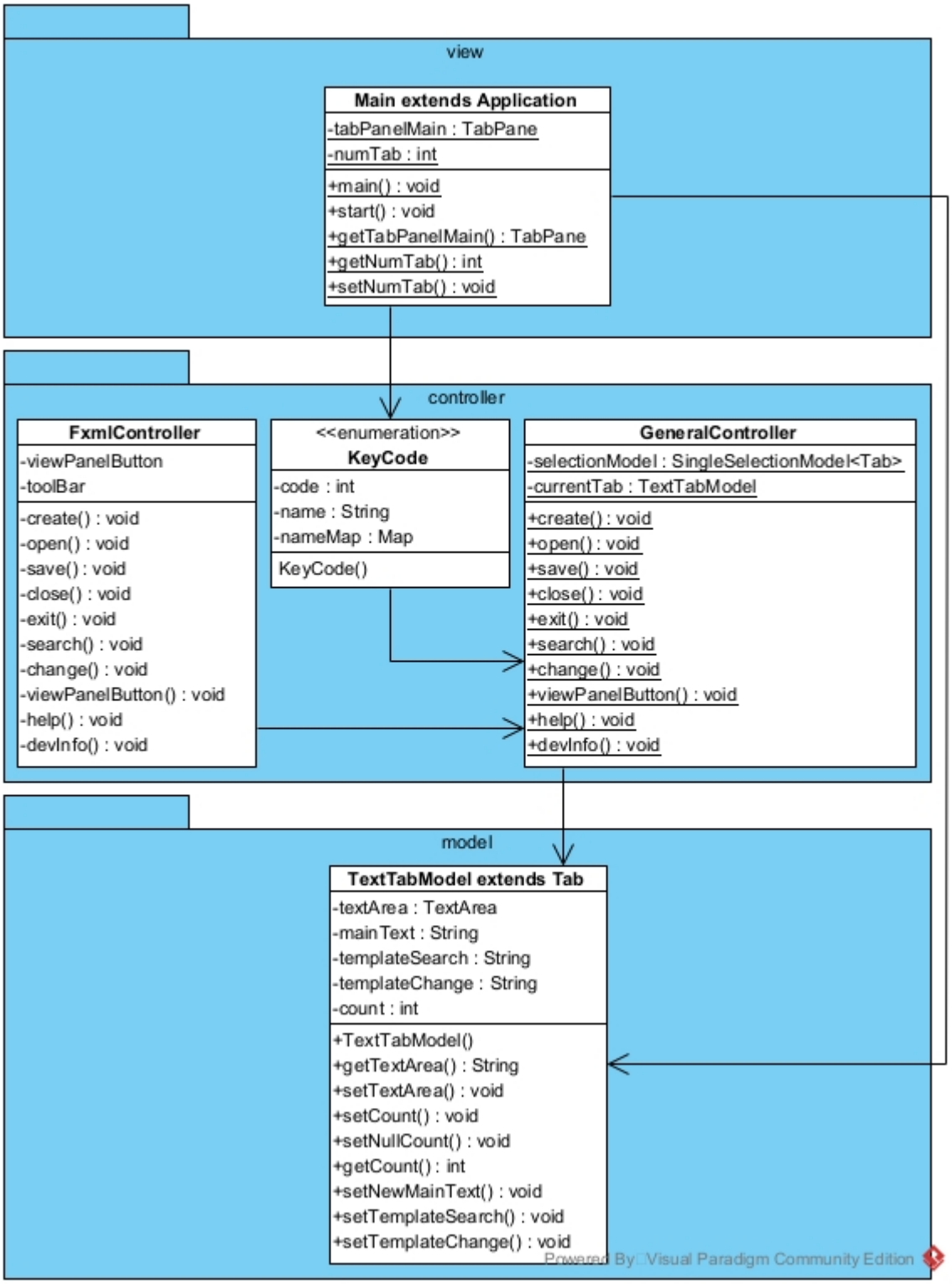

Рисунок 3.1 – Диаграмма классов

4. ОПИСАНИЕ КЛАССОВ

При написании данного приложения использовались 5 классов, для размещения которых с целью структурирования файлов приложения и удобства его разработки и сопровождения использовались следующие папки-пакеты: *model*, *view* и *controller*.

Класс *Main* (*package main.view*) – основной стартовый класс приложения. Также является основным классом-представителем компонента представления. *Представление* (View) отвечает за отображение данных предметной области (*модели*) пользователю, реагируя на изменения *модели*.

Класс *Main* состоит из двух полей и 5 методов.

К полям относятся следующие:

- *tabPanelMain* – является графическим элементом типа *TabPane*; центральный элемент графического интерфейса, представляющий из себя панель вкладок;
- *numTab* – является счетчиком открытых вкладок, использующийся для изменения имени вкладки при создании новой вкладки.

К методам относятся следующие.

1. Основной метод выполнения приложения *main()* – расширяет класс Application.

2. Основной метод, отвечающий за построение визуального интерфейса приложения – *start()*. При этом структура приложения извлекается из файла *Main.fxml*, стиль отображения – из файла *main.css*.

Также в данном методе задаются имя приложения в рамке окна («Блокнот»), иконка приложения в рамке, стартовый размер приложения, стартовое расположение приложения на экране (по центру).

Кроме этого, подключается прослушиватель событий сочетания горячих клавиш для управления приложением с клавиатуры (впоследствии все клавиши будут синхронизированы с кнопками визуального интерфейса).

3. Метод-геттер *getTabPanelMain()* – используется классом GeneralController для управления вкладками приложения.

4. Метод-геттер getNumTab() – используется для получения номера, необходимого для заголовка при создании новой вкладки.

5. Метод-сеттер getNumTab() – используется для изменения (увеличения) номера, необходимого для заголовка при создании новой вкладки.

Классы FmxlController, GeneralController и KeyCode (*package main.controller*) являются классами-представителями компонента контроллера. В данном случае все классы *контроллера* (Controller) участвую в интерпретации действия пользователя, оповещая *модель* о необходимости изменений.

Класс *FmxlController* состоит из двух полей и 10 методов.

Все 10 методов по сути являются обработчиками событий при нажатии на кнопки и пункты меню (визуальные компоненты интерфейса приложения, определенные в классе *Main.fxml*).

Каждый из методов в свою очередь вызывает соответствующий метод в классе *GeneralController*, который по факту и является не только основным классом-контроллером, определяющим необходимые действия приложения, но и классом, синхронизирующим действия горячих клавиш, определенных в классе *Main*, и действия элементов интерфейса файла *Main.fxml*.

При этом два поля *toolBar* и *viewPanelButton* необходимы для передачи в составе метода *viewPanelButton()* в класс *GeneralController*.

Класс *GeneralController* также состоит из двух полей и 10 методов.

Оба поля (*selectionModel* и *currentTab*) являются глобальными переменными класса, которые неоднократно используются в методах класса. Поля представляют из себя модель выбора вкладок на панели (только одна вкладка в текущий момент времени) и собственно – текущую вкладку.

К методам относятся следующие.

1. Метод создания новой вкладки *create()*.
2. Метод открытия текстового файла *.txt *open()*.
3. Метод сохранения текстового файла *.txt *save()*.

4. Метод закрытия текущей вкладки *close()*.

5. Метод выхода из приложения (закрытия) *exit()*.

6. Метод открытия окна поиска *search()*.

7. Метод открытия окна замены *change()*.

8. Метод скрытия/отображения элемента инферфейса панели кнопок *viewPanelButton()*.

9. Метод открытия окна помощи *help()*, структурно реализованного с помощью файла *help.fxml*, стилистически определенного с помощью файла *help.css*.

10. Метод открытия окна *devInfo()*, содержащего сведения о разработчике.

Класс *KeyCode* является enum-классом, являющимся локальной реализацией класса *javafx.scene.input.KeyCode*, и содержащим сведения только о клавишах, использующихся в приложении в качестве горячих.

Класс *TextTabModel* (*package main.model*) – основной класс-представитель компонента модели. Модель (Model) предоставляет данные предметной области представлению и реагирует на команды контроллера, изменяя свое состояние.

Класс *TextTabModel* состоит из 5 полей, конструктора и 8 методов.

Данный класс расширяет стандартный JavaFX класс Tab. Расширение стандартного класса было выполнено таким образом, чтобы каждый документ класса представлял из себя не просто отдельную вкладку, а отдельный текстовый документ с возможностью редактирования, поиска и замены текста.

Для реализации указанных функций были введены 5 переменных:

1) textArea – текстовая область для ввода текста;
2) mainText – основной текст, содержащийся в текстовой области;
3) templateSearch – шаблон (символ, слово или текст) для поиска;
4) templateChange – шаблон (символ, слово или текст) для замены;

5) count – счетчик для подсчета количества совпадений при поиске или количества произведенных замен.

Конструктор TextTabModel(String fileName) принимает текстовое поле *fileName*, которое генерируется в методе *GeneralController.create()* при создании новой вкладки или в методе *GeneralController.open()* при открытии уже существующего текстового файла и добавляется в качестве заголовка в создаваемую конструктором вкладку.

Также в конструкторе уже существующей переменной textArea присваивается значение нового текстового поля, устанавливается размер шрифта, текстовое поле добавляется на создаваемую вкладку, после чего на текстовое поле перемещается фокус курсора.

К методам относятся:

1) getTextArea() – метод-геттер поля textArea (при сохранении, поиске и замене текста);
2) setTextArea() – метод-сеттер поля textArea (при открытии файла);
3) setCount() – метод-сеттер поля count (при поиске и замене);
4) setNullCount() – метод-сеттер поля для приведения к нулевому значению после выполненного поиска или замены;
5) getCount() – метод-геттер (при поиске и замене);
6) setNewMainText() – метод-сеттер для передачи в текстовое поле нового текста после замены;
7) setTemplateSearch() – метод-сеттер для передачи шаблона поиска из поля окна в экземпляр модели;

setTemplateChange() – метод-сеттер для передачи шаблона замены из поля окна в экземпляр модели.

5. ВЛИЯНИЕ ВЫБРАННЫХ АРХИТЕКТУРНЫХ СТИЛЕЙ НА СТРУКТУРУ КЛАССОВ

В результате структура разрабатываемого приложения является отражением как классического шаблона проектирования «Model-View-Controller», так и объектно-ориентированной архитектуры.

Указанная структура находит свое отражение в диаграмме классов как в виде разработанных классов, так и в виде визуального объединения классов или их групп в рамках того или иного пакета (папки-пакеты («package»): *model*, *view* и *controller*).

Список использованных источников

1. Милихин М.М. Проектирование и архитектура программных средств: методические указания по выполнению лабораторных работ для студентов ФДО направления подготовки 09.03.04 «Программная инженерия». Томск: ФДО, ТУСУР, 2017. 14 с.

2. Милихин М.М. Проектирование и архитектура программных средств: учебное пособие. Томск: факультет дистанционного обучения ТУСУРа, 2015. 138 с.