МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра игровой индустрии (ИИ)

ОТЧЁТ ПО УЧЕБНОЙ ПРАКТИКЕ

Ознакомительная практика

приказ Университета о направлении на практику от «13» февраля 2024 г. №1457-С

Отчет представлен к
рассмотрению:
Студент группы ИКБО-30-23 «__» июня 2024 Мусатов И.А.
(подпись и расшифровка подписи)

Отчет утвержден.
Допущен к защите:

Руководитель практики
от кафедры «__» июня 2024 Шутов К.И.
(подпись и расшифровка подписи)

Москва 2024 г.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра игровой индустрии (ИИ)

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ
Ознакомительная практика

Студенту 1 курса учебной группы ИКБО-30-23
Мусатову Ивану Алексеевичу

Место и время практики: РТУ МИРЭА кафедра ИИ, с 09 февраля 2024 г. по 31 мая 2024 г.

Должность на практике: студент

1. СОДЕРЖАНИЕ ПРАКТИКИ:

1.1. Изучить: архитектурный паттерн MVC, теорию разработки Desktop приложений.

1.2. Практически выполнить: разработать архитектуру Desktop приложения с применением паттерна MVC и реализовать ее при помощи объектно-ориентированного языка программирования Java и соответствующих библиотек графической визуализации.

1.3. Ознакомиться: с разработкой программных продуктов на языке Java и с инструментами создания графического интерфейса.

2. ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ: подготовить доклад на научно-техническую конференцию студентов и аспирантов РТУ МИРЭА или иную конференцию, подготовить презентационный материал.

3. ОРГАНИЗАЦИОННО-МЕТОДИЧЕСКИЕ УКАЗАНИЯ: в процессе практики рекомендуется использовать периодические издания и отраслевую литературу годом издания не старше 5 лет от даты начала прохождения практики.

Руководитель практики от кафедры
«09» февраля 2024 г. _____ (Шутов К.И.)
Подпись

Задание получил
«09» февраля 2024 г. _____ (Мусатов И.А.)
Подпись

СОГЛАСОВАНО:
Заведующий кафедрой:

«09» февраля 2024 г. _____ (Зуев А.С.)
Подпись

Проведенные инструктажи:

Охрана труда: «09» февраля 2024 г.

Инструктирующий _____ Зуев А.С., зав. кафедрой ИИ
Подпись

Инструктируемый _____ Мусатов И.А.
Подпись

Техника безопасности: «09» февраля 2024 г.

Инструктирующий _____ Зуев А.С., зав. кафедрой ИИ
Подпись

Инструктируемый _____ Мусатов И.А.
Подпись

Пожарная безопасность: «09» февраля 2024 г.

Инструктирующий _____ Зуев А.С., зав. кафедрой ИИ
Подпись

Инструктируемый _____ Мусатов И.А.
Подпись

С правилами внутреннего распорядка ознакомлен: «09» февраля 2024 г.

_____ Мусатов И.А.
Подпись

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

РАБОЧИЙ ГРАФИК ПРОВЕДЕНИЯ ОЗНАКОМИТЕЛЬНОЙ ПРАКТИКИ

студента Мусатова И.А. 1 курса группы ИКБО-30-23 очной формы обучения, обучающегося по направлению подготовки 09.03.04 Программная инженерия.

Неделя	Сроки выполнения	Этап	Отметка о выполнении
1	09.02.2024	Подготовительный этап, включающий в себя организационное собрание (Вводная лекция о порядке организации и прохождения производственной практики, инструктаж по технике безопасности, получение задания на практику)	
5	09.03.2024	Подготовительный этап (Участие в круглом столе на тему «Проблема достоверности информации в современном мире»)	
5	10.03.2024	Подготовительный этап (Участие в круглом столе на тему «Информационно-коммуникационные технологии для организации информационного	
8	29.03.2024	Подготовительный этап (Участие в круглом столе на тему «Информационная и библиографическая культура»)	
8	30.03.2024	Исследовательский этап (Поиск информации об архитектурном паттерне MVC; анализ материала по применению языка Java в разработке Desktop приложений с элементами графической визуализации)	
11	19.04.2024	Представление руководителю структурированного материала: теоретический анализ архитектуры приложения	
11	20.04.2024	Технологический этап (Разработка векторного графического редактора на языке Java с использованием требуемых библиотек)	
15	17.05.2024	Представление созданного программного продукта руководителю практики	

15	18.05.2024	Согласование с руководителем доклада на научно-техническую конференцию студентов и аспирантов РТУ МИРЭА	
16	31.05.2024	Подготовка окончательной версии отчета и программного продукта (Оформление материалов отчета в полном соответствии с требованиями на оформление письменных учебных работ студентов)	

Руководитель практики от кафедры _____/Шутов К.И., ассистент/

Обучающийся _____/Мусатов И.А./

Согласовано:

Заведующий кафедрой _____/Зуев А.С., к.т.н., доцент/

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

При создании любого приложения неизбежным вопросом для разработчика становится структура программной реализации. Именно она определяет дальнейшую логику проектирования программного продукта, а также используемые в процессе разработки средства. Возникает необходимость в структуризации и формализации самого процесса разработки. Использование определенной логики проектирования при создании ПО позволяет упростить ориентирование в коде и придает данному коду абстрагированное представление, семантику.

В мировой практике проектирования приложений уже созданы и четко выверены некоторые архитектурные паттерны. Один из них, MVC, будет рассмотрен и применен на практике в процессе выполнения данной работы.

Целью данной ознакомительной практики является применение архитектурного подхода к разработке приложения.

Перед автором работы стоят следующие задачи, решение которых рассмотрено в соответствующих разделах отчета:

- Изучить теоретические основы архитектурного паттерна MVC;
- Ознакомиться с языком программирования Java и библиотекой графического интерфейса JavaFX;
- Разработать Desktop приложение в соответствии с архитектурой.

Навык формализованной разработки, безусловно, является полезным подспорьем в профессиональной деятельности разработчика.

1 РАССМОТРЕНИЕ АРХИТЕКТУРНОГО ПАТТЕРНА MVC

1.1 История создания MVC

В 1970х годах зарождалась структуризация разработки в ее современном понимании. Если на уровне программной реализации она выражалась в появлении парадигмы структурного программирования, сформулированной Эдсгером Дейкстрой, то на архитектурном уровне новым видением стал подход MVC.

Создателем нового паттерна проектирования выступил Трюгве Ренскауг (Trygve Reenskaug). В качестве приглашенного ученого он работал в исследовательском центре Xerox Palo Alto Research Center над созданием объектно-ориентированного языка программирования Smalltalk-79. В процессе работы ему понадобился шаблон, который мог бы применяться для структурирования любой программы, где пользователям необходимо взаимодействовать с большим объемом информации.

Изначально шаблон должен был состоять из четырех частей: Model, view, thing и editor. Но при обсуждении этой структуры с другими разработчиками из Smalltalk было принято решение использовать только три части: model, view и controller.

В окончательной версии шаблона разработчики рассматривали модель как сугубо отдельную часть программы. View извлекало данные из model для их отображения пользователю. При этом view обеспечивало и обратную связь между пользователем и model. Controller использовался как организационный элемент пользовательского интерфейса, который координировал работу нескольких view, получал сигналы от пользователя и передавал их своим view.

Эта версия также включала в себя некий editor, особый вид контроллера, который использовался для редактирования внешнего представления.

Именно этот вариант и лег в основу языка программирования Smalltalk-80. В общем плане, view представляло множество вариантов отображения информации пользователю; controller предлагал варианты взаимодействия с пользователем; model же была тесно связана с view, но ее реализация производилась полностью на усмотрение разработчика. Среда разработки Smalltalk-80 также включала в себя инструмент "MVC Inspector", позволяющий наглядно представлять структуру описанных model, view и controller.

В 1988 году два бывших сотрудника PARC выпустили статью в журнале The Journal of Object Technology, где представили MVC как парадигму программирования и методологию для разработчиков на Smalltalk-80. Тем не менее, их реализация отличалась как от идеи Ренскауга, так и от версии из справочников Smalltalk-80. В их версии view обеспечивала любую работу с графическим интерфейсом, в то время как controller становился более абстрактным объектом, получающим сигналы пользователя и взаимодействующим с одной моделью и несколькими представлениями.

Развитие сетевых технологий в 1990е годы неизбежно повлекло за собой создание web-приложений. Структуризация разработки была необходима и в этой сфере, в связи с чем велась работа над интерпретацией уже существующего шаблона MVC.

В 1996 году в связи с появлением ПО WebObjects, которое было написано на Objective-C, принципы MVC стали активно популяризироваться. После интеграции WebObjects с Java шаблон стал распространен и среди разработчиков на Java. Эта тенденция была поддержана и впоследствии выпускаемыми фреймворками.

В 2003 году Мартин Фаулер (Martin Fowler) опубликовал работу "Patterns of Enterprise Application Architecture", в которой MVC представлен как шаблон, где controller получает запрос, отправляет соответствующие сообщения объекту

модели, принимает ответ от объекта модели и передает его соответствующему представлению для отображения.

Различные фреймворки, такие как Ruby on Rails и Django стали использовать аналогичный подход к интерпретации MVC. С тех пор этот шаблон проектирования позиционируется как средство быстрого развертывания (rapid deployment), что позволяет ему быть широко распространенным и за пределами корпоративной среды.

Шаблон MVC стал очень удобным и понятным в применении средством проектирования архитектуры приложения, поэтому, применительно к различным контекстам, из него произошли и некоторые другие шиблоны. Например, иерархический model–view–controller (HMVC), model–view–adapter (MVA), model–view–presenter (MVP), model–view–viewmodel (MVVM) и другие.

1.2 Архитектура MVC

Как уже было упомянуто выше, паттерн MVC состоит из трех частей: модели – model, представления – view и контроллера – controller. Рассмотрим подробнее устройство и назначение каждой из них.

1.2.1 Модель

Модель – это центральный компонент паттерна, предоставляющий данные и методы их обработки. Модель не зависит ни от внешнего представления приложения, ни от контроллера, так как ее основной задачей является описание данных, логики и правил функционирования приложения. При этом модель может изменять свое состояние в процессе работы. Также не исключено наличие нескольких представлений у одной и той же модели.

1.2.2 Представление

Представление (или вид) – это любое графическое отображение информации пользователю, будь то таблица, диаграмма или график. Отвечает только за формирование пользовательского интерфейса, не поддерживает обработку данных, передаваемых пользователем.

1.2.3 Контроллер

Контроллер – это прослойка, обеспечивающая связь между пользователем и внутренним представлением. Контроллер отслеживает все входные данные и конвертирует их в команды для модели или представления. В некоторых вариациях может выступать лишь как прослойка между представлением и моделью, в таком случае контроллер существует в единственном экземпляре для каждого представления. В более распространенном случае могут существовать несколько контроллеров, каждый из которых привязан к определенной модели. При этом сама модель может принимать сигналы от нескольких контроллеров.

1.2.4 Обобщение теории

Паттерн MVC позволяет разделить логические части приложения на три независимо создаваемых блока. При этом изменение любого из них не влечет за собой изменение остальных. Можно, например, для одного и того же внутреннего представления написать несколько вариантов интерфейса. Или менять алгоритмы обработки данных без изменения внешнего представления. Схематичное изображение логики работы паттерна представлено на Рисунке 1.1.

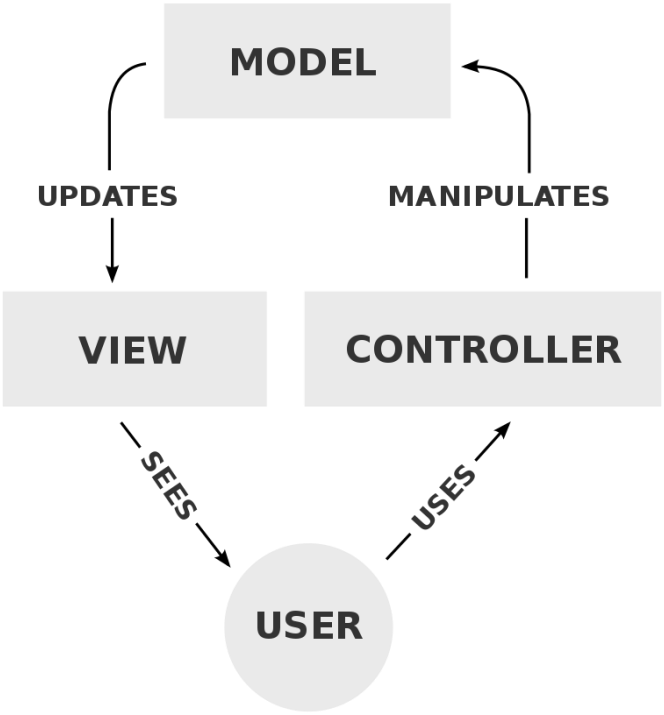

Рисунок 1.1 – Условная схема архитектурной логики MVC

Стоит также отметить, что архитектура MVC не имеет строгой реализации. В каждом фреймворке она может быть реализована по-своему. Тем не менее, существует некоторый стандартный набор наиболее удобных шаблонов. Например, паттерн «Наблюдатель» для отслеживания событий – сигналов. Практические приемы еще будут рассмотрены в данной работе.

1.3 Рассмотрение ошибок использования паттерна

Отсутствие строго определенной структуры паттерна порождает возможность его неправильной трактовки и использования. Рассмотрим наиболее распространенную ошибку понимания принципов работы архитектуры MVC.

Ошибочным является трактовка MVC как пассивной архитектурной модели. В таком рассмотрении контроллер берет на себя большую часть задач

модели, становясь носителем бизнес-логики приложения. Модель же в таком случае превращается в примитивную совокупность функций доступа к данным.

В среде разработчиков подобная реализация получила яркое наименование: «Толстые, тупые, уродливые контроллеры» (Fat Stupid Ugly Controllers).

В действительности же MVC – активная архитектурная модель. Модель здесь не только выступает в качестве средства доступа к данным, но и содержит в себе всю бизнес-логику функционирования приложения. Контроллер же является лишь элементом информационной системы и ответственен только за прием, анализ и передачу запроса от пользователя. Только такое понимание позволяет реализовать контроллер как прослойку в соответствии с изначальной идеей архитектуры MVC.

2 РАЗРАБОТКА ПРИЛОЖЕНИЯ

2.1 Выбор языка программирования и IDE

Разработка приложений с продуманной архитектурой предполагает использование усовершенствованных парадигм программирования. Наиболее удобной парадигмой для обеспечения как логики работы приложения, так и взаимодействия с пользователем, является объектно-ориентированная.

Существует большое количество языков программирования, поддерживающих объектно-ориентированную парадигму. Каждый из них имеет свои области применения и специализацию. Мы же рассмотрим наиболее известных и широко применяемых представителей – C++ и Java. Проведем их небольшую сравнительную характеристику и определимся с выбором языка. Краткие результаты анализа представлены в Таблице 2.1.

Таблица 2.1 – Таблица сравнения языков программирования

Критерий сравнения	**Java**	**C++**
Простота разработки	+	-
Транслятор	Виртуальная Java-машина	Компилятор
Скорость запуска приложения	-	+
Кроссплатформенность	+	-
Разнообразие встроенных библиотек	+	-

Основным различием данных языков является способ трансляции в машинный код. Если у C++ это компилятор, то у Java есть специальная виртуальная машина, которая переводит исходный код в байт-код. С одной стороны, такая особенность Java увеличивает время запуска приложений, написанных на ней. С другой стороны, это обеспечивает кроссплатформенность приложения. Если говорить о преимуществах Java, то стоит также отметить

наличие поддерживаемых и удобных библиотек графической визуализации. Например, Swing и JavaFX. Также, по личному мнению автора, процесс разработки на Java несколько проще, чем на C++.

Для проектирования приложения в рамках данной работы нет требования по максимальной эффективности продукта. Основная задача работы – на практике применить архитектурный паттерн MVC в процессе разработки. С учетом данной оговорки считаем применение языка Java оправданным ввиду удобства для разработчика.

Существует множество IDE (Integrated Development Environment) для разработки на Java. Пожалуй, наиболее удобной и продвинутой является IntelliJ IDEA, разработанная компанией JetBrains. IntelliJ IDEA имеет встроенные алгоритмы анализа кода, облегчающие ориентирование в коде и помогающие разработчику с выполнением рутинных действий. Среда открыта к интеграции со сторонними фреймворками и другими языками программирования. Также IntelliJ IDEA позволяет работать с Git – распределенной системой управления версиями.

2.2 Рассмотрение библиотеки графического интерфейса

Для создания графического интерфейса на платформе языка Java существуют три больших фреймворка. Это AWT, Swing и JavaFX. Swing представляет собой более усовершенствованную версию AWT, поэтому ограничим наше сравнение библиотеками Swing и JavaFX.

Библиотека Swing, начиная с 1998 года, входит в состав JRE (Java Runtime Environment), т.е. не нуждается в дополнительной загрузке. JavaFX же, начиная с 11ой версии Java, больше не входит в состав JDK (Java Development Kit), поэтому для создания интерфейса на JavaFX требуется дополнительное скачивание данной платформы. Тем не менее, если рассматривать историю

возникновения и развития данных библиотек, то JavaFX была создана в 2008 году в качестве замены для Swing. JavaFX предоставляет более обширный набор компонентов пользовательского интерфейса, например, позволяет работать с 2D и 3D-графикой, использовать технологии стилизации, такие как CSS и FXML. Стоит также отметить кроссплатформенность JavaFX. Если Swing может быть использован только для создания Desktop-приложений, то JavaFX поддерживает возможность Web-разработки и разработки мобильных приложений. К тому же JavaFX, особенно ввиду ее отделения от Oracle, имеет растущее сообщество разработчиков и регулярно получает обновления и улучшения. Таким образом, видим, что JavaFX – более современное и удобное средство создания графического интерфейса.

2.3 Техническое описание приложения

Как уже было отмечено ранее, целью данной работы является осмысленное использование архитектуры MVC в процессе создания Desktop-приложения. Поэтому ценность самого программного продукта не является первоочередной задачей.

Определим задачу разработки следующим образом: требуется разработать Desktop-приложение, предоставляющее пользователю возможность работы с примитивами векторной графики. Необходимо обеспечить удобное взаимодействие с пользователем при помощи оконного интерфейса, а также возможность представления результата работы с векторной графикой в виде распространенного формата SVG (Scalable Vector Graphics).

2.4 Определение структуры приложения

Приступим к непосредственному составлению архитектуры приложения. Так как среди задач данного приложения есть взаимодействие с пользователем, то проектирование стоит начать с пользовательского интерфейса.

Любая среда редактирования предполагает взаимодействие в том числе и через панель меню. Поэтому окно нашего приложения должно содержать шапку меню. Для работы непосредственно с векторной графикой потребуется холст, на котором будут отрисовываться объекты, создаваемые пользователем. Для его масштабируемости необходимо использовать прокручиваемую область. Для выбора инструментов взаимодействия с холстом понадобится панель с выбором инструментария. Также необходимым средством станет работа с цветовой палитрой. Ее можно реализовать через еще одну панель, содержащую инструменты выбора цвета.

В теоретической части данной работы было установлено, что контроллер являет собой прослойку между представлением и моделью. Для каждого представления создается свой контроллер. Наше окно для работы с векторной графикой также должно иметь своего обработчика событий. Потребуется рассматривать взаимодействие пользователя с элементами меню, с панелью инструментов, панелью выбора цветов, а также обрабатывать взаимодействие с холстом. То есть, необходимо отслеживать следующие действия пользователя на холсте: обработка сигналов от мыши и от клавиатуры.

Векторная графика предполагает работу с графическими элементами как с объектами. То есть существует определенный инструментарий для создания фигур. Основной задачей модели нашего приложения будет хранение и обработка созданных графических объектов, а также их интерпретация в формате SVG.

Описанная структура отображает общую архитектуру приложения. Конкретная реализация может потребовать создания дополнительных вспомогательных интерфейсов, например, диалоговых окон.

2.5 Работа над пользовательским интерфейсом

Существует два способа разработки пользовательского интерфейса на платформе JavaFX. Первый – это описание иерархии и свойств элементов интерфейса непосредственно через создание объектов в Java-коде. Второй же способ – использование поддерживаемого формата FXML (FX Markup Language). FXML – это декларативный язык разметки пользовательского интерфейса на основе XML. Данный способ позволяет явным образом абстрагировать проектирование представления от логики программы. Подобный вариант разработки помогает предотвратить смешение понятий представления и модели. То есть, способствует использованию паттерна MVC. Ввиду этого считаем целесообразным разработку пользовательского интерфейса при помощи FXML.

Для создания дизайна воспользуемся графическим редактором SceneBuilder. SceneBuilder – это отдельный инструмент визуального макетирования, который позволяет разрабатывать пользовательский интерфейс без написания кода. То есть, созданный в SceneBuilder макет автоматически переводится в FXML-файл.

Реализуем пользовательский интерфейс основного окна, структуру которого мы рассмотрели в предыдущем пункте. Корневым объектом в иерархии элементов является панель Pane. В нее для удобства последующего размещения элементов положим панель BorderPane. Эта панель позволяет разделять пространство на пять частей: верх, низ, лево, право и центр. В верхнюю часть определим панель меню MenuBar. В левую – панель инструментов ToolBar. В

нижнюю – другую панель ToolBar для работы с цветом. В центральную же часть поместим прокручиваемую область ScrollPane, в которой расположим холст Canvas. Реализация интерфейса при помощи SceneBuilder представлена на Рисунке 2.1.

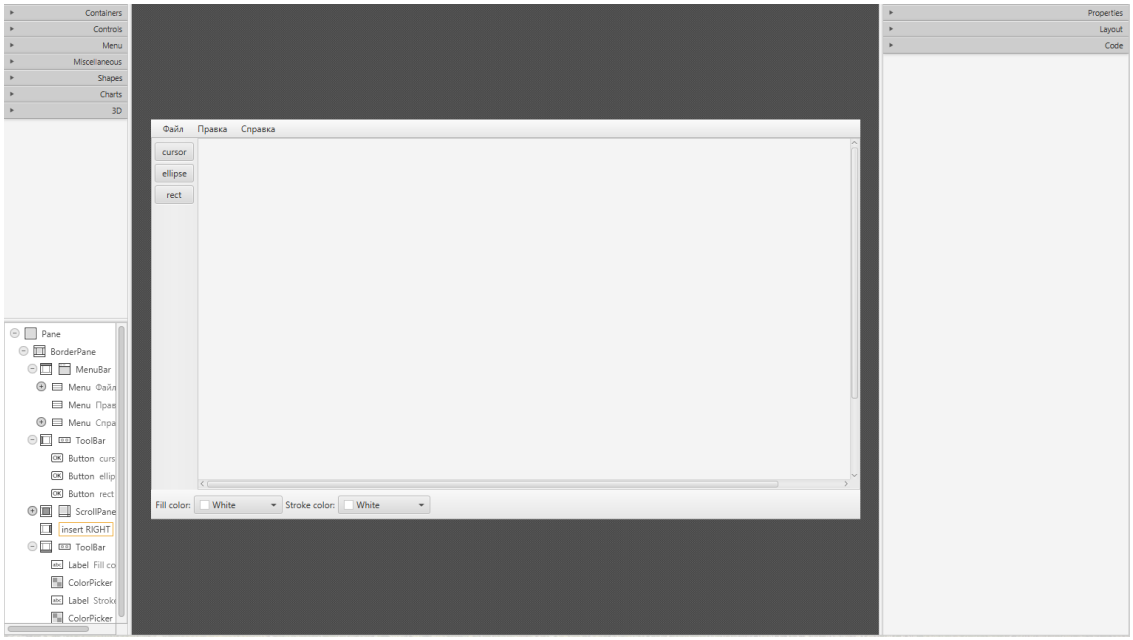

Рисунок 2.1 – Интерфейс главного окна приложения в SceneBuilder

Отметим также, что для взаимодействия с пользователем помимо основного окна будут использоваться вспомогательные окна выбора файлов. В частности, диалоговое окно открытия файла и диалоговое окно сохранения файла. Создание данных окон обеспечивается вызовами методов showOpenDialog и showSaveDialog встроенного в JavaFX класса FileChooser.

2.6 Разработка модели приложения

В паттерне MVC модель отвечает как за логику работы приложения, так и за хранимые данные. В нашем случае данными, над которыми осуществляются операции, являются объекты векторной графики. В рамках данной работы определим следующие объекты, называемые фигурами: прямоугольник и эллипс.

Для описания функционала работы с фигурой создадим абстрактный класс AbstractFigure. Данный класс содержит следующие поля и методы:

- cFill – приватное поле, хранящее цвет заливки фигуры;
- cStroke – приватное поле, хранящее цвет контура фигуры;
- strokeWidth – приватное поле, хранящее толщину контура;
- Стандартные методы get_ и set_ (геттеры и сеттеры) для каждого из имеющихся приватных полей;
- draw – абстрактный метод отрисовки фигуры;
- isInArea – абстрактный метод проверки наличия точки с данными координатами в области, занимаемой данной фигурой;
- displace – абстрактный метод смещения центра фигуры на заданные приращения координат;
- SVGSerialize – абстрактный метод перевода характеристик фигуры в формат представления svg-элемента.

От данного абстрактного класса наследуются два других класса: Rectangle и Ellipse. Рассмотрим описание каждого из них. Начнем с класса Rectangle:

- x – приватное поле, содержащее x-координату левого верхнего угла прямоугольника;
- y – приватное поле, содержащее y-координату левого верхнего угла прямоугольника;
- width – приватное поле, хранящее длину прямоугольника;
- height – приватное поле, хранящее ширину прямоугольника;
- Стандартные методы get_ и set_ (геттеры и сеттеры) для каждого из имеющихся приватных полей;
- Rectangle – стандартный конструктор с установкой всех имеющихся полей объекта;
- Перегруженные методы базового класса AbstractFigure.

Описание класса Ellipse:

- cx – приватное поле, содержащее x-координату центра эллипса;
- cy – приватное поле, содержащее y-координату центра эллипса;

- rx – приватное поле, хранящее радиус эллипса по x;
- ry – приватное поле, хранящее радиус эллипса по y;
- Стандартные методы get_ и set_ (геттеры и сеттеры) для каждого из имеющихся приватных полей;
- Ellipse – стандартный конструктор с установкой всех имеющихся полей объекта;
- Перегруженные методы базового класса AbstractFigure.

После программной реализации представления объектов векторной графики можно перейти к разработке логики функционирования модели.

Отметим, что для работы с графикой пользователю доступно три инструмента: курсор, прямоугольник и эллипс. Поведение модели будет меняться в зависимости от выбора этих состояний. Поэтому для удобства определим перечисляемый тип данных State, который может принимать одно из следующих значений: CURSOR, RECT и ELLIPSE.

Для слаженной работы с множеством создаваемых фигур необходимо иметь определенную функциональную коллекцию, которая хранила бы и обрабатывала их всех. Создадим класс Model, который будет наследником коллекции ArrayList из стандартного пакета java.util. Типом данных, хранящихся в Model, будет AbstractFigure. Организованный нами полиморфизм позволяет работать с абстрактными фигурами в целом, не уточняя их тип. Класс Model будет также отвечать и за логику функционирования приложения, поэтому он должен иметь связь с контроллером.

Отдельно необходимо отметить, что, ввиду однородности обрабатываемых данных, модель у нашего приложения единственная. Поэтому применительно к данному классу можно реализовать паттерн программирования «Одиночка» (Singletone). Это означает, что в процессе работы программы будет создан только единственный экземпляр данного класса, который, вообще говоря, можно хранить как поле самого класса.

Перейдем к описанию полей и методов класса Model:

- instance – приватное статическое поле, хранящее экземпляр данного класса;
- controller – приватное поле, хранящее объект класса MainController, необходимого для установки связи с контроллером;
- cFill – приватное поле, хранящее цвет заливки, выбираемый на панели выбора цвета;
- cStroke – приватное поле, хранящее цвет контура, выбираемый на панели выбора цвета;
- state – публичное поле перечисляемого типа данных State, характеризующее состояние системы;
- path – публичная переменная для хранения пути к файлу, в который записывается результат работы с графикой;
- Стандартные методы get_ и set_ (геттеры и сеттеры) для каждого из имеющихся приватных полей;
- getInstance – статический метод получения экземпляра данного класса. Реализация паттерна Singleton;
- draw – метод отрисовки всех элементов текущей коллекции;
- getFigureByPoint – метод получения фигуры, в область которой попадает данная точка;
- deleteFigure – метод удаления фигуры из коллекции;
- convertToSVG – метод перевода всех характеристик созданного рисунка в формат svg.

Рассмотренные выше классы позволяют организовать независимую от внешнего представления логику функционирования приложения, что и соответствует требованиям архитектуры MVC.

2.7 Разработка функционала контроллеров приложения

После независимого друг от друга определения представления и модели, можем перейти к реализации связующей прослойки – контроллера.

При реализации пользовательского интерфейса в формате FXML необходимо было указать класс-контроллер, обрабатывающий сигналы. Назовем этот класс MainController. Опишем его поля и методы:

- canvas – приватное поле, ссылающееся на объект canvas из внешнего представления;
- fillColor – приватное поле, ссылающееся на объект fillColor из панели выбора цвета во внешнем представлении;
- strokeColor – приватное поле, ссылающееся на объект strokeColor из панели выбора цвета во внешнем представлении;
- stage – приватное поле, хранящее ссылку на окно внешнего представления;
- listeners – приватное поле-коллекция типа ArrayList, хранящая объекты типа UserListener;
- MainController – стандартный конструктор класса;
- setStage – метод, устанавливающий ссылку на главное окно внешнего представления;
- cursorSet – метод установки состояния модели на CURSOR;
- ellipseSet – метод установки состояния модели на ELLIPSE;
- rectSet – метод установки состояния модели на RECT;
- canvasMousePressed – метод отслеживания зажимания кнопки мыши;
- canvasMouseReleased – метод отслеживания отпускания кнопки мыши;
- canvasMouseDragged – метод отслеживания смещения курсора;
- globalKeyPressed – метод отслеживания нажатия клавиши на клавиатуре;

- setFillColor – метод отслеживания смены цвета в панели fillColor;
- setStrokeColor – метод отслеживания смены цвета в панели strokeColor;
- canvasRepaint – метод отрисовки графики на холсте canvas;
- menuNew – метод отслеживания элемента меню «создать»;
- menuOpen – метод отслеживания элемента меню «открыть»;
- menuSave – метод отслеживания элемента меню «сохранить»;
- menuSaveAs – метод отслеживания элемента меню «сохранить как»;
- menuExit – метод отслеживания элемента меню «выйти»;
- menuAbout – метод отслеживания элемента меню «справка».

Обработка перемещения объектов-фигур на холсте требует временного хранения рассматриваемой фигуры. Для того, чтобы не усложнять описание класса MainController, создадим вспомогательные делегируемые контроллеры-слушатели. Для реализации удобного полиморфизма сначала определим классы-интерфейсы MouseListener и KeyboardListener, в которых опишем прототипы функций, реагирующих на активность пользователя при помощи мыши и клавиатуры соответственно. Методы класса MouseListener:

- mouseClicked – метод реагирования на клик кнопки мыши;
- mousePressed – метод реагирования на зажатие кнопки мыши;
- mouseReleased – метод реагирования на отпускание кнопки мыши;
- mouseDragged – метод реагирования на перемещение мыши.

Методы класса KeyboardListener:

- keyPressed – метод реагирования на зажатие клавиши клавиатуры;
- keyReleased – метод реагирования на отпускание клавиши клавиатуры;
- keyTyped – метод реагирования на набор клавиши клавиатуры.

Унаследуем два данных класса новым классом-интерфейсом UserListener. Определим в нем еще два метода:

- resetFill – метод реагирования на смену цвета заливки;
- resetStroke – метод реагирования на смену цвета контура.

На основе базового класса UserListener определим производные классы наших делегируемых слушателей. Виды слушателей совпадают с состояниями системы: CursorListener, RectListener и EllipseListener. По очереди опишем каждый из них. Поля и методы класса CursorListener:

- x – приватное поле для фиксации x-координаты точки холста;
- y – приватное поле для фиксации y-координаты точки холста;
- chosenFigure – приватное поле, хранящее рассматриваемую фигуру класса AbstractFigure;
- updateFigure – метод обновления местоположения рассматриваемой фигуры;
- Перегруженные методы базового класса UserListener.

Поля и методы класса RectListener:

- x – приватное поле для фиксации x-координаты точки привязки прямоугольника;
- y – приватное поле для фиксации y-координаты точки привязки прямоугольника;
- rect – приватное поле, хранящее создаваемый прямоугольник класса Rectangle;
- updateRect – метод обновления отрисовки создаваемого прямоугольника;
- Перегруженные методы базового класса UserListener.

Поля и методы класса EllipseListener:

- x – приватное поле для фиксации x-координаты точки центра эллипса;
- y – приватное поле для фиксации y-координаты точки центра эллипса;
- ellipse – приватное поле, хранящее создаваемый эллипс класса Ellipse;
- updateEllipse – метод обновления отрисовки создаваемого эллипса;
- Перегруженные методы базового класса UserListener.

Созданные классы-слушатели позволят основному контроллеру MainController делегировать сигналы от пользователя, связанные с рисованием.

Подобная реализация позволит разгрузить функционал MainController и обеспечит расширяемость набора объектов векторной графики.

2.8 Анализ работы приложения

На предыдущих этапах постепенно реализовывались все структурные элементы приложения. Соединим их и проанализируем результат.

При запуске приложения появляется окно, с которым пользователь может взаимодействовать через следующие элементы: шапка меню, инструменты для рисования, панель выбора цветов. К каждому из этих элементов подвязан метод контроллера. Если действия пользователя связаны с созданием графического изображения на холсте, то контроллер обращается к модели и обеспечивает ее корректное функционирование. Изменившаяся модель заставляет контроллер передавать холсту новую отрисовку, т.е. происходит опосредованное влияние модели на представление.

Пример работы в созданном приложении изображен на Рисунке 2.2. Случай работы с диалоговым окном изображен на Рисунке 2.3.

Схематичная визуализация архитектуры разработанного приложения представлена в Приложении А.

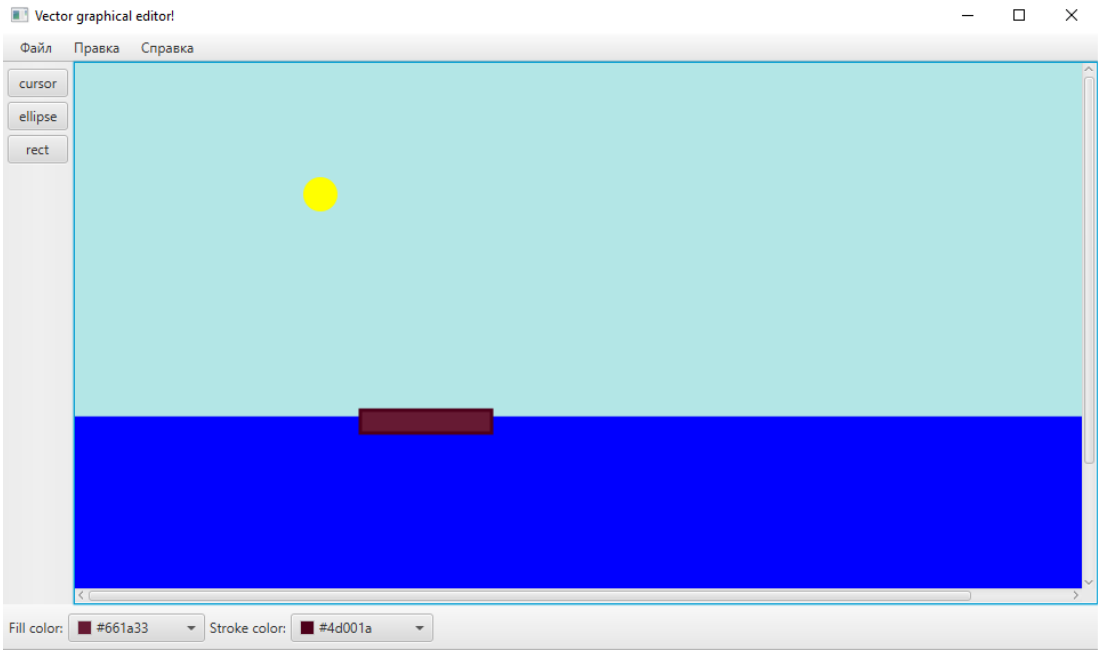

Рисунок 2.2 – Пример работы в реализованном приложении

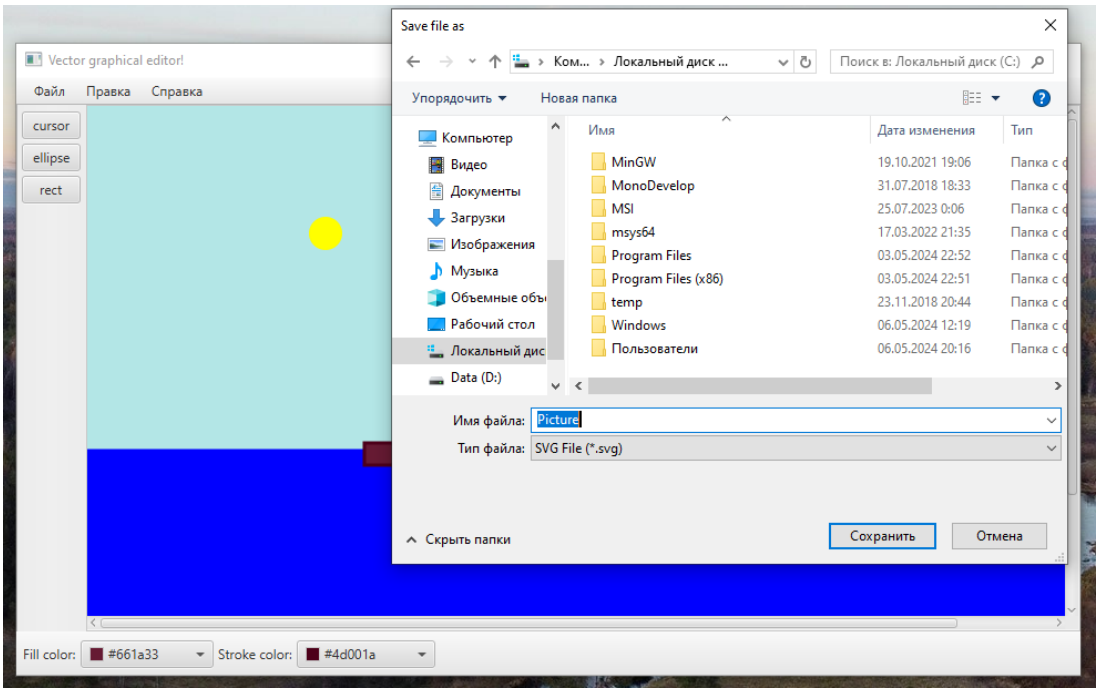

Рисунок 2.3 – Пример работы с диалоговым окном

ЗАКЛЮЧЕНИЕ

В процессе выполнения работы был теоретически изучен архитектурный паттерн MVC. Рассмотрена история возникновения и развития данной архитектуры. При помощи информации из открытых источников проанализированы характеристики компонентов паттерна. Отдельному рассмотрению подлежали распространенные ошибки проектирования.

На основе изученной информации был применен осмысленный архитектурный подход к разработке приложения. В соответствии с составленными техническими требованиями сформирована теоретическая структура приложения, удовлетворяющая паттерну MVC.

Для реализации программного продукта выбран язык программирования Java, а также фреймворк JavaFX для графической визуализации. При помощи средств разработки структура приложения реализована на практике. Функционирование продукта протестировано и признано удовлетворительным.

В результате ознакомительной практики автор приобрел опыт построения архитектуры Desktop-приложения, ознакомился с современным фреймворком JavaFX и научился тестировать программный продукт. Полезным также стал опыт работы с иностранными источниками. Приобретенные знания и навыки, безусловно, способствуют развитию профессиональных качеств в рамках специальности «Программная инженерия».

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Krasner, G.E. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80/G.E. Krasner, S.T. Pope // Journal of Object-Oriented Programming. - 1988.-Vol. 1.-PP. 26-49.
2. Trygve Home Page: [Электронный ресурс]. URL: https://folk.universitetetioslo.no/trygver/index.html (Дата обращения: 13.05.2024)
3. Обобщенный Model-View-Controller: [Электронный ресурс]. URL: https://rsdn.org/article/patterns/generic-mvc.xml (Дата обращения: 13.05.2024)
4. Java AWT vs Java Swing vs Java FX: [Электронный ресурс]. URL: https://www.geeksforgeeks.org/java-awt-vs-java-swing-vs-java-fx/ (Дата обращения: 13.05.2024)
5. Client Technologies: Java Platform, Standard Edition (Java SE) 8: [Электронный ресурс]. URL: https://docs.oracle.com/javase/8/javase-clienttechnologies.htm (Дата обращения: 13.05.2024)
6. Учебник по JavaFX: начало работы: [Электронный ресурс]. URL: https://habr.com/ru/articles/474292/ (Дата обращения: 13.05.2024)
7. Руководство по JavaFX: [Электронный ресурс]. URL: https://metanit.com/java/javafx/ (Дата обращения: 13.05.2024)

Приложение А

Схема архитектуры приложения

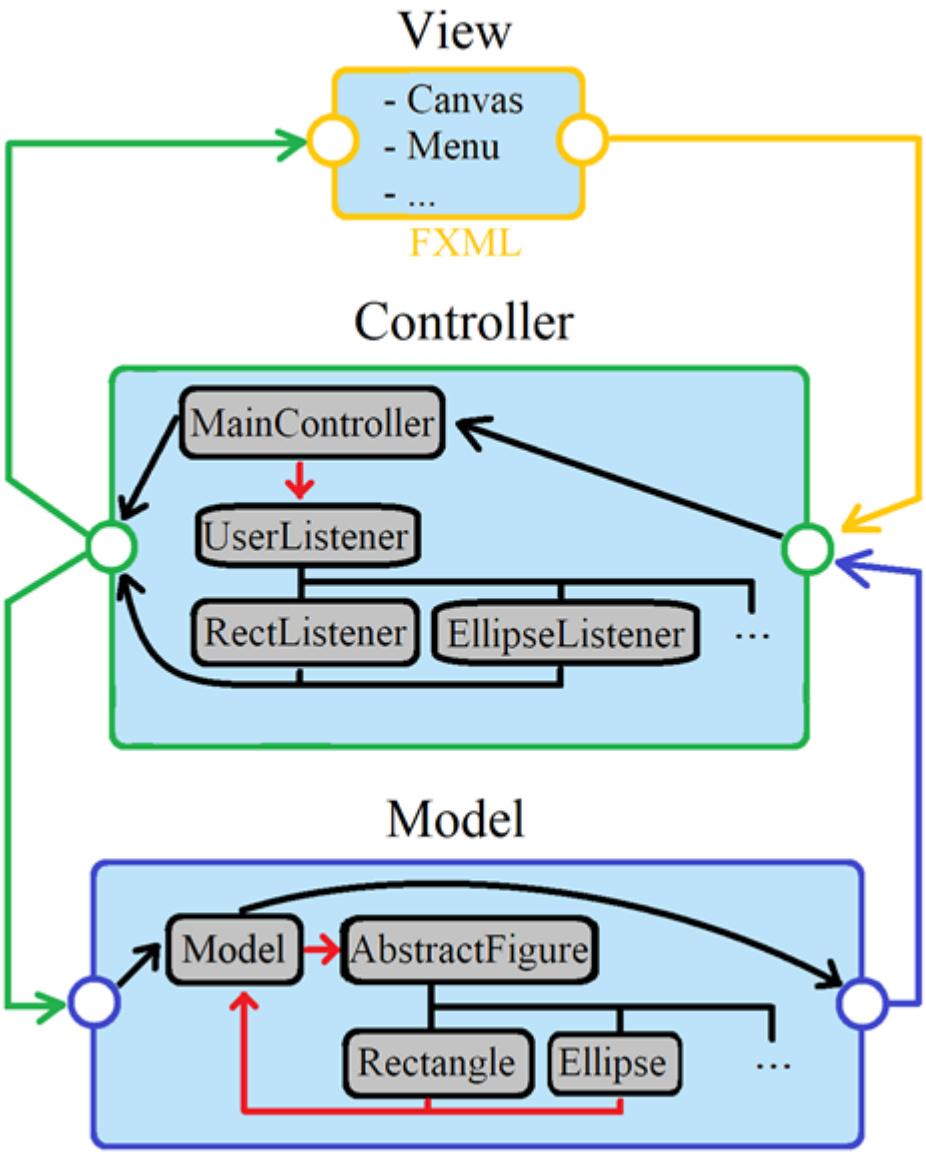

Рисунок А.1 – Условная схема архитектуры приложения