**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 6](#_Toc121825855)

[1 ОБЗОР ПРЕДМЕТНОЙ ОБЛОСТИ 7](#_Toc121825856)

[1.1 Подходы к формированию экзаменационных билетов 7](#_Toc121825857)

[1.2 Обзор аналогов. Обзор существующих подходов и методик 8](#_Toc121825858)

[2 ПОСТАНОВКА ЗАДАЧИ 12](#_Toc121825859)

[3 ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ АЛГОРИТМОВ 15](#_Toc121825860)

[3.1 Алгоритм выборки вопросов, группированных по сложности 15](#_Toc121825861)

[3.2 Алгоритм выборки вопросов, группированных по темам 17](#_Toc121825862)

[3.3 Алгоритм «сплошного списка» 18](#_Toc121825863)

[4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 20](#_Toc121825864)

[4.2 Схема программного обеспечения 21](#_Toc121825865)

[4.3 Описание основных классов программы 23](#_Toc121825866)

[4.4 Тестовый пример и руководство пользователя 27](#_Toc121825867)

[ЗАКЛЮЧЕНИЕ 37](#_Toc121825868)

[ПРИЛОЖЕНИЕА*.* ИСХОДНЫЙКОДПРИЛОЖЕНИЯ 39](#_Toc121825869)

[ПРИЛОЖЕНИЕВ*.* ТРЕБОВАНИЕ К СТРУКТУРНОМУ СОДЕРЖИМОМУ ФАЙЛА ВОПРОСОВ 48](#_Toc121825870)

# ВВЕДЕНИЕ

Быть продуктивным, успевать эффективно выполнять максимальное количество важных дел в установленные сроки – это то, к чему стремиться каждый человек в любой сфере жизни. И тут нам никак не справиться без автоматизированных систем, которые бы выполнили работу за нас.

И так, в университетах, с целью оценивания знаний студентов, качество усвоения материала по изучаемым дисциплинам, проводят экзамены. Для провидения экзаменов каждому преподавателю, по окончанию текущего семестра, приходиться сталкиваться с составлением экзаменационных билетов, составление которых является важной частью учебно-методической деятельности преподавателя любого учебного заведения. Для любого предмета имеющего теоретическую основу требуется составление экзаменационных вопросов и упражнения для проверки знания полученных студентом во время курса образования. Для того чтобы правильно проэкзаменовать студента необходимо чтобы подборка вопросов была случайно и разной для каждого студента, при этом не исключена вероятность в необходимости контролировать количество повторения вопросов, а также формировать билеты, вопросы которых группированы по сложности или темам. Поэтому качественное составление экзаменационных билетов представляет собой сложный и трудоёмкий процесс. Так, для провидения экзамена в среднем разрабатывается по 30 билетов каждый из которых содержит по 3 – 4 вопроса. Эта задача трудоемка, так как требует от составителя определенную степень внимательности, аккуратности и достаточного количество времени, дабы соблюсти требования по их составлению. Намного удобнее было если бы преподаватель имел какую-то систему, которая бы позволила за короткие строки и усилия, предоставив список вопросов, внести данные в шаблон билета и получить решение рутинной задачи.

Поэтому целью проекта было создание Desktop-ого приложения «Оптимизация составления экзаменационных билетов при их автоматизированном формировании», предназначенная для преподавателей с целью облегчения процесса составления бланка билетов из готовых списков вопросов, и предоставить различные варианты обработки входных вопросов, указав сложность, число возможного повторения и тему вопросов, а так же их оптимизированной выборке при генерация билетов.

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛОСТИ

Соблюдение всех стандартов в ведении документации является  
неотъемлемым условием внедрения политики качества в любой сфере  
деятельности. Одним из важнейших документов, которые должны быть  
предоставлены преподавателем по правилам приема экзамена, наряду с  
рабочей программой является комплект экзаменационных билетов. Сам  
билет должен соответствовать определенным стандартам, и оформляется по определенному шаблону, который состоит из названия учреждение образование, факультета, кафедры, специальности студентов, вид сессии, список вопросов, ФИО экзаменатора и заведующего кафедрой, а также дату утверждения и номер протокола.

Обычно задача по составлению и хранению билетов решается при  
помощи текстового процесса, при помощи программы «Microsoft Office» или  
«OpenOffice.org». В этом случае комплект билетов представляет собой  
документ данного текстового процессора, на каждой странице которого  
размещаются один или два экземпляра билета. При большом объеме материалов поддерживать в актуальном состоянии и, тем более, вносить  
изменения в ручную, становится для преподавателя непростой задачей, требующей от него значительных затрат времени и сил.

Как правило, вначале преподаватель составляет список вопросов по всем темам, пройденным с учениками. Обычно их бывает от 30 до 70, но может быть и больше. Перечень контрольных вопросов в сумме должен охватывать основной материал учебной программы. Такой список нужно отдать ученикам заблаговременно, а на основе него составить экзаменационные билеты.

Следующим этапом является формирования самих билетов, поэтому для начала определяется количество всего билетов, и количество вопросов в одном билете, а после приступают к заполнение шаблонных данных и к распределения вопросов [1].

**1.1 Подходы к формированию экзаменационных билетов**

Многие преподаватели распределяют задания по какому-нибудь алгоритму, например, берут вопросы с разных концов списка, делят по темам или по сложности и добавляют в билеты по одному заданию из каждого раздела как в последовательном, так и в произвольном порядке.

Главное правило при составлении билетов — это принцип справедливого распределения вопросов в них, без перекосов, то есть чтобы во все билеты были равны по сложности. Часто бывает, что в билете три вопроса: *легкий, сложный* и *очень* *сложный*. Однако, бывают и такие случаи, что в билет попадают либо все сложные вопросы, либо все лёгкие вопросы, так как выбор вопросов в билет часто производится случайно. Поэтому распределение вопросов должно следовать какой-то системе, исключающей беспорядочное размещение вопросов в билете. Бессистемность приводит к тому, что билеты становятся не­равноценными по важности вопросов и не дают возможности судить о том, как подготовлен студент к наиболее важным сторонам предстоящей ему работы [1].

С другой стороны, часто бывает случаи, что в билете все вопросы примерно одинаковой важности и Вам необходимо сгенерировать билеты так, чтобы в билете было обязательно три вопроса из разных тем, например *вопрос 1* – это вопросы из темы «Динамическое программирование», *вопрос 2* – вопросы из темы «Базовые понятия методов оптимизации» *вопрос 3* – вопросы из темы «Многокритериальная оптимизация». Так же не исключена вероятность, что преподаватель просто решит распределить вопросы в произвольном порядке.

В любом случаи, помимо всего вышеперечисленного, перед преподавателем стоит задача их равномерной выборки и распределению в билете, удерживая во внимании повторяемость вопроса. Грубо говоря преподаватель решает задачу оптимизации, целью которой является достичь максимальной релевантности всех билетов, дабы справедливо оценить знания студентов по окончанию учебного курса, что очень трудоёмка при ручном способе составления, и часто приводит к путанице. В принципе, каждый такой метод распределения достаточно обоснован, поэтому только самому преподавателю остается решать, каким воспользуется он [2].

## 1.2 Обзор аналогов. Обзор существующих подходов и методик

Ручной способ составления билетов, является самым распространенным, однако не единственным способом формирования бланка билетов, при том, что первый требует больших временных так и человеческих затрат, и усилий, по сравнению с уже имеющимися на сегодняшний день сервисами, позволяющие выполнить ту же задачу за короткий промежуток времени. Однако на данный момент существует не так много систем для решение поставленной задачи. Большинство из них существуют в виде веб-сайтов, а за приобретение некоторых требуется внести плату. Рассмотрим некоторые из них.

**1.2.1 Веб приложение «Создание билетов для зачета или экзамена по спискам вопросов различных типов»**

Данное веб-приложение обладает следующими достоинствами [4]:

1. доступно через браузер;
2. легкодоступный, не требующий денежных затрат;
3. возможность получить результат в виде файла;
4. возможность сгруппировать вопросы по категории;

И следующие недостатки:

1. ограничение на число вопросов в билете: максимум 3;
2. отсутствует определения шаблона для билета;
3. ограничение на количество строк, содержащийся в одном вопросе;
4. отсутствует какое-либо возможность форматирование текста вопроса;
5. отсутствует возможность ввода различных графических изображений и математических формул;
6. не контролируется повторяемость вопросов в случаи их нехватки.
7. неудобный пользовательский интерфейс;

Главное окно данного веб-приложения приведено на рисунке 1.1.

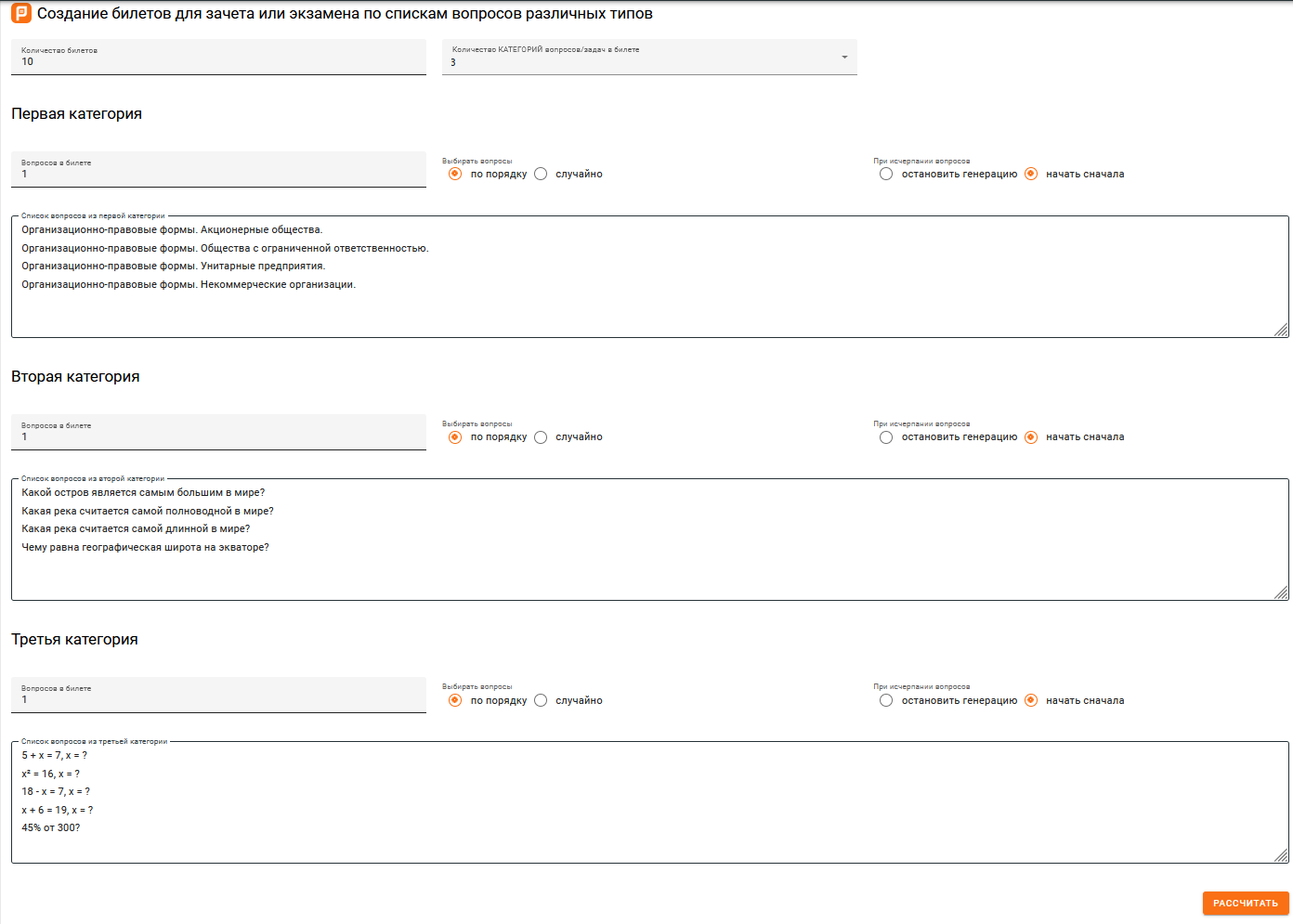
****

Рисунок 1.1 – Главное окно веб-приложения

**1.2.2 Desktop приложение «Генератор экзаменационных билетов»**

Данное приложение обладает следующими преимуществами:

1. присутствует База Данных, позволяющая добавлять, редактировать, удалять введенные вопросы;
2. возможность сгруппировать вопросы по сложности;
3. возможность выбрать тему вопросов.
4. простой пользовательский интерфейс.
5. получить результат в виде файла;
6. присутствует определения шаблона для билета учреждение образования.

И следующие недостатки:

1. ограничение на число вопросов в билете: максимум 6;
2. отсутствует какое-либо возможность форматирование текста вопроса;
3. отсутствует возможность ввода различных графических изображений и математических формул;
4. не контролируется повторяемость вопросов в случаи их нехватки;
5. приложения является платным.

Данное приложение содержит следующие окна, приведенные на рисунках 1.2 – 1.4:

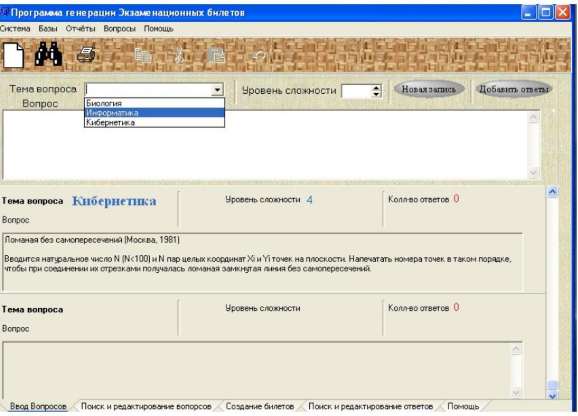


Рисунок 1.2 – Окно для ввода в базу данных вопрос.

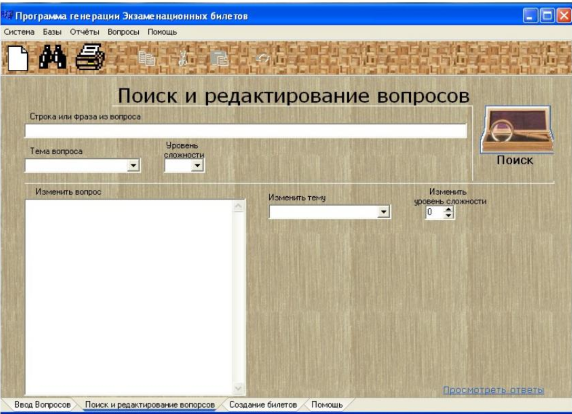


Рисунок 1.3 – Окно редактирования вопросов

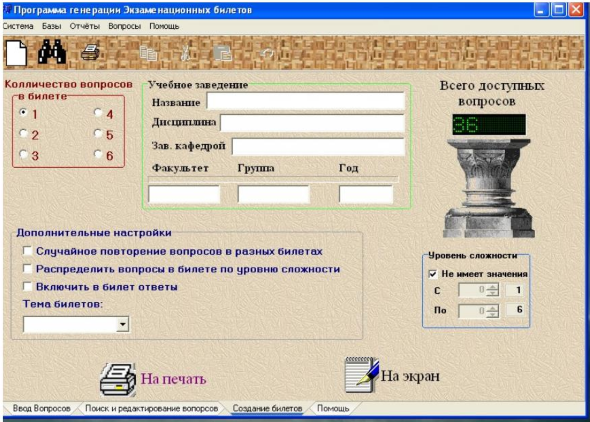
****

Рисунок 1.4 – Окно создания билетов

2 ПОСТАНОВКА ЗАДАЧИ

Обзор аналогов для решение поставленной задачи, по генерацию билетов, выполненных в разделе 1, позволяет сделать вывод, что приведенные аналоги обладают своими особенностями, не позволяющие полностью решить вопрос формирования бланков экзаменационных билетов. Для роботы некоторых сервисов требуется доступ в интернет, а в случаи необходимости иметь больше чем три вопроса в билете, ввода математических формул, графических изображений, а также контролировать повторяемость вопросов, аналоги не позволяют сделать этого.

Поэтому тема курсового проекта «Оптимизация составления экзаменационных билетов при их автоматизированном формировании» актуальна.

Целью данного проекта является создание приложения, обеспечивающего широкий спектр вариативности при составлении билетов.

*Исходными данными*: является список экзаменационных вопросов, хранящиеся в файле формата MS Word, имеющую заданную структуру оформления. Требование, к структурному содержимому файла вопросов смотрите **приложение В**.

*Выходными данными:* является готовый бланк билетов, представленный файлом формата MS Word.

Приложение должно предоставлять достаточную гибкость в выборе метода составления билетов, позволяющего распределить вопросы как в сплошном порядке, так и сгруппировать вопросы по сложности или темам; позволять преподавателю иметь широкий спектр возможностей по форматированию содержимого вопроса и управлять их повторяемостью. К задаче разрабатываемого приложения так же входит учёт оптимизированной выборки вопросов из списка, при составления билетов, сохраняя способность приложения функционировать с той же автономностью и с достаточной производительностью. Так же при необходимости размещения двух и более билетов на одном листе А4 предусмотреть возможность настройки этот параметр.

Главное условие при составлении экзаменационных билетов заключается в том, чтобы соблюсти равномерное распределение и выборку вопросов, чтобы каждый сгенерированный билет был уникален среди уже имеющихся, если это возможно, либо уменьшить вероятность появления билетов с одинаковыми вопросами. Для этого необходимо выполнить следующие задачи:

1. определить алгоритмы оптимизационного группирования экзаменационных вопросов в билетах:

* группирование по сложности;
* группирование по темам;
* алгоритм «сплошного списка»

1. выполнить программную реализацию:

* определить средства разработки;
* спроектировать схему ПО;
* спроектировать основные алгоритмы;
* провести тестирование;
* составить руководство пользователя;

Обзор проблемной области, перечень задач курсового проектирования и позволяет создать диаграмму последовательностей и прецедентов, представленных на рисунках 2.1 – 2.2. Задача так же состоит разработка удобного и интуитивно понятного интерфейса пользователя.

*Диаграмма прецедентов (Use case diagram)* описывает функциональные требования системы с точки зрения прецедентов и включает в себя два компонента — участника (Actor) и прецедент (Use case). Прецедент описывает отдельный случай поведения системы с точки зрения пользователя.

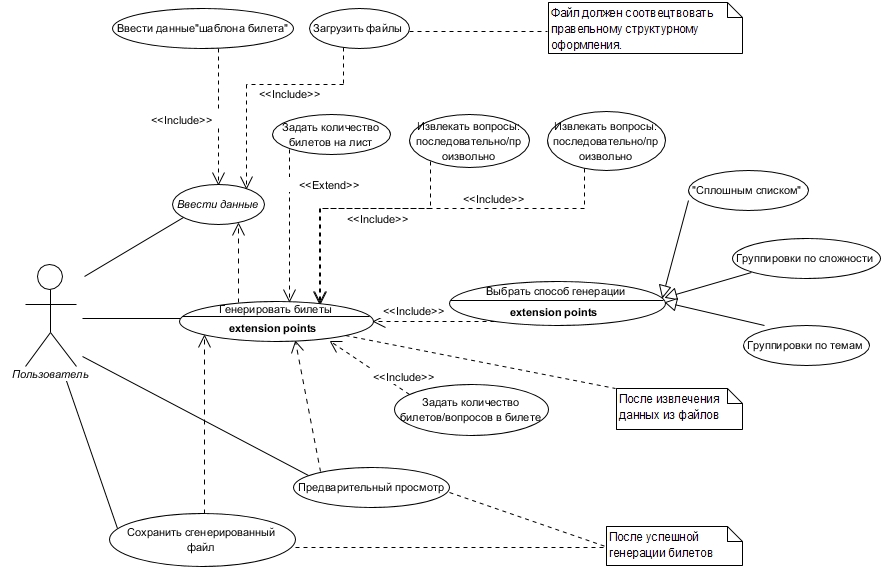


Рисунок 2.1 – Диаграмма прецедентов использования приложения

*Диаграмма последовательности* создает модель взаимодействия объектов. Основой такого взаимодействия служит временная последовательность, которая дает представление о взаимодействии объектов в конкретном прецеденте. отражает взаимодействие объектов в динамике, во времени.

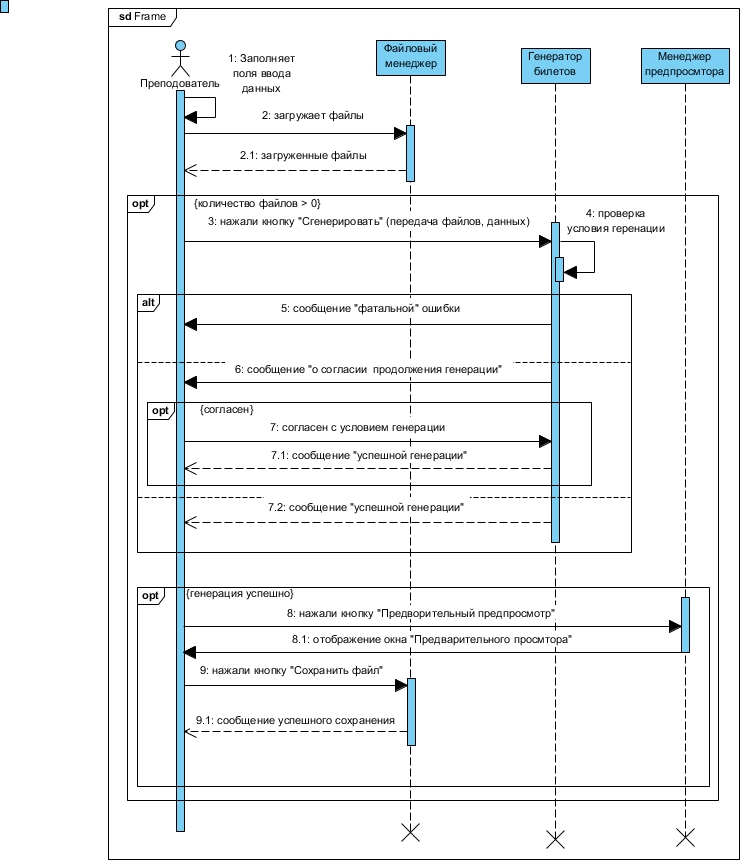
****

Рисунок 2.2 – Диаграмма последовательности использования приложения

3 ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ АЛГОРИТМОВ

В курсовом проекте три алгоритма формирования экзаменационных билетов:

* алгоритм выборки вопросов, сгруппированных по сложности;
* алгоритм выборки вопросов, сгруппированных по темам;
* алгоритм «сплошного списка»

Каждый из представленных алгоритмов так же должен выполнять задачу оптимизации, которая заключается в том, чтобы соблюсти равномерное распределение и выборку вопросов, чтобы каждый сгенерированный билет был уникален среди уже имеющихся, если это возможно, либо уменьшить вероятную возможность появления билетов с одинаковыми вопросами.

Ниже описанные *алгоритмы формирования билетов* позволяет управлять *порядком считывания/записи* вопросов: последовательно либо произвольно.

По умолчанию вопросы считываются из источников-вопросов *последовательно*, однако можно задать и *произвольное*. В случаи *произвольного* считывания вопросов, мы произведем простую перетасовку вопросов в источнике-вопросов – действие над источником-вопросов.

По умолчанию вопросы сформированного билета, будут записаны *последовательно*, то есть в том порядке в каком они были сформированы в результате действия *алгоритма выборки вопроса из списка*, однако этот порядок можно нарушить, задав *произвольную* запись простой перетасовкой вопросов в билете – действие над билетом.

## 3.1 Алгоритм выборки вопросов, группированных по сложности

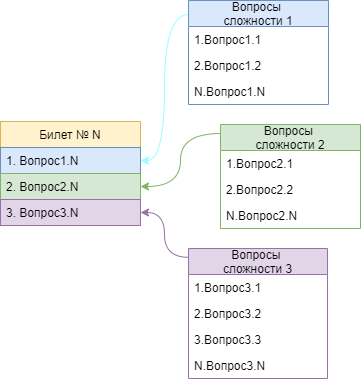
*Условие выполнения алгоритма:* входные вопросы должен быть сгруппированы по сложности в виде списков. Каждый такой список соответствует номеру вопроса в билете. Будут рассматриваться только те списки, сложность которых входят в диапазон [1; количествоВопросовБилета], в случаи отсутствие какого-либо из них, алгоритм не выполняется. Требуется, чтобы количество вопросов в каждом списке-сложности суммарно был равен как минимум требуемому количеству билетов с учётом указанного количества повторения указанных вопросов. Пример представлен на рисунке 3.1.

Рисунок 3.1 – Формирование билетаОписание алгоритма формирования билетов приведено на рисунке 3.2.

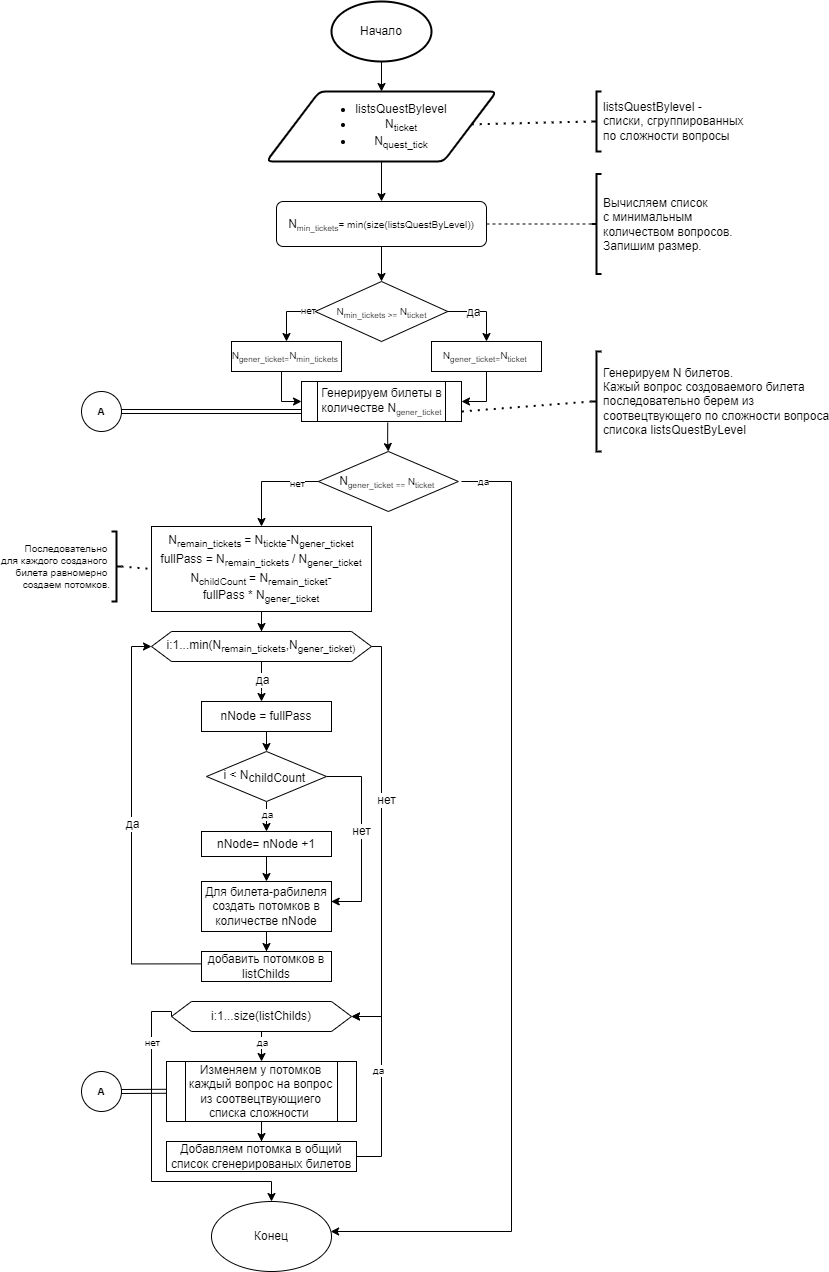
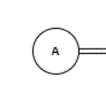


Рисунок 3.2 – Блок-схема алгоритма формирования билета

Далее опишем алгоритм, *описывающий выборку вопроса из списка*, сложность которого соотносимо с текущим вопросом создаваемого билета. Применение этого алгоритма приведено на рисунке 3.2 под блок-схемой:

Алгоритм произведем на примере одного из таких списков, вопрос которого будут извлекаться в соответствующий вопрос создаваемого билета эквивалентной сложности. Описание алгоритма приведено на рисунке 3.3.

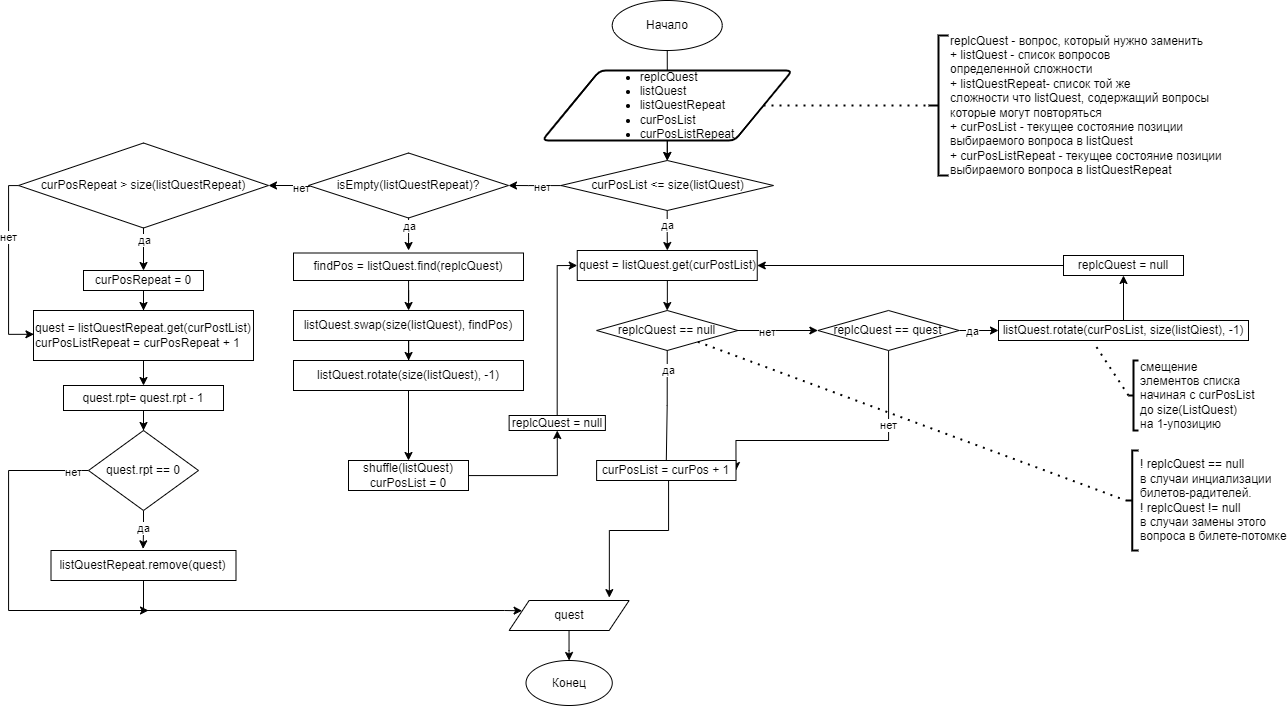


Рисунок 3.3 – Блок-схема алгоритма выборки вопроса из списка

## 3.2 Алгоритм выборки вопросов, группированных по темам

*Условие выполнения алгоритма:* входные вопросы должен быть сгруппированы по темам в виде списков. Каждый такой список соответствует номеру вопроса в билете. Требуется, чтобы количество тем было равно как минимум количеству вопросов в билете. Эта группировка схожа с *алгоритмом выборки вопросов, группированных по сложности*, только в место сложностей: 1, 2, 3, …, N, здесь используется Тема1, Тема2, …, ТемаN. Примеры алгоритмов представлен на рисунках 3.1 – 3.3.

## 3.3 Алгоритм «сплошного списка»

*Условие выполнения алгоритма:* количество входных вопросов, должно быть равно как минимум количеству вопросов в одном билете. Требуется чтобы количество вопросов суммарно с учётом указанного количество повторения вопросов был равен как минимум:

Пример алгоритма представлен на рисунке 3.4.

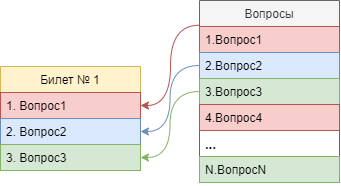


Рисунок 3.4 – Представление формирования билетов

Алгоритм формирования билетов схож с блок-схемой, описанной ранее, на рисунке 3.1, однако входной параметр *listsQuestByLevel,* в данному случаи будет является простым список, считаный с файлов. Здесь следует описать алгоритм, *описывающий выборку вопроса из списка.* Описание алгоритма представлено на рисунке 3.5.

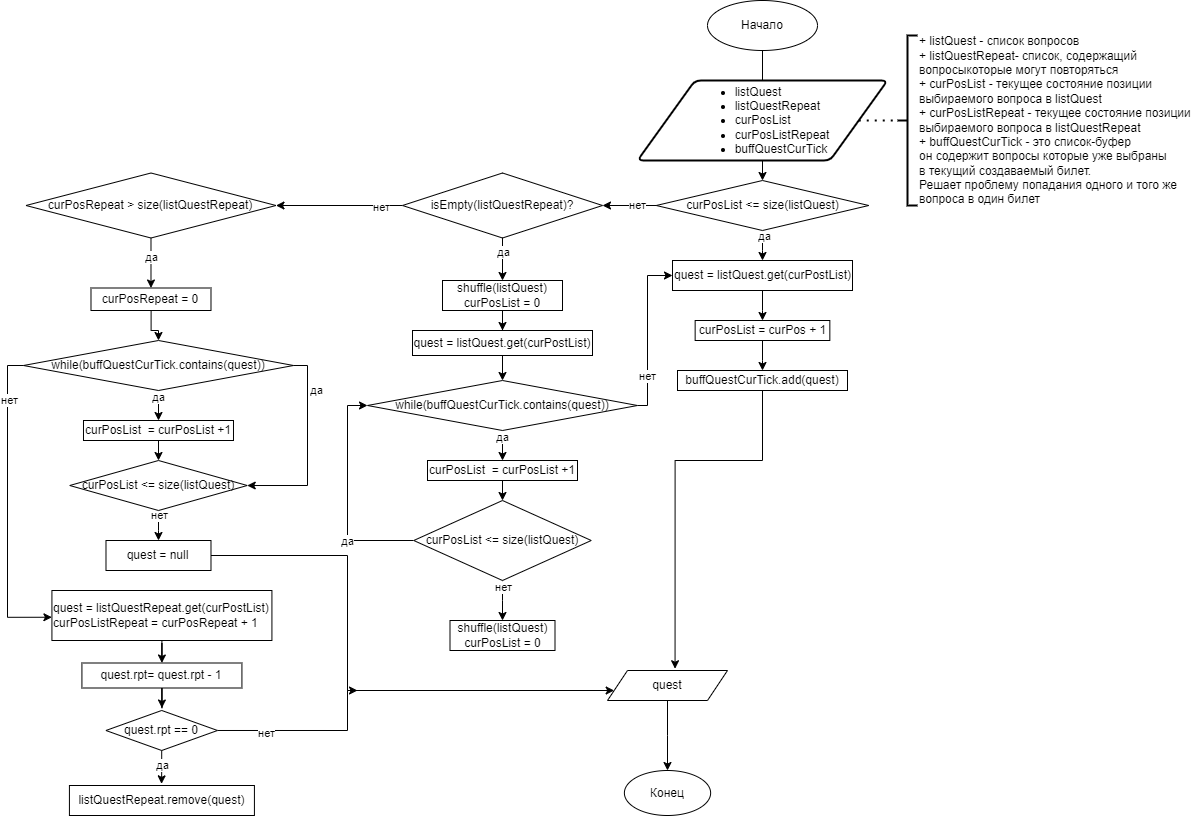


Рисунок 3.5 – Блок-схема алгоритма вопроса из списка

4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

**4.1 Обзор средств разработки**

Приложение для данного проекта создано с использованием языка программирования Java версии 17 и его графических библиотек AWT и Swing, а так библиотеки с открытым кодом FlatLaf и SoftSmithy Utility Library Swing. В качестве инструмента сборки использован Meven. Для парсинга файла Microsoft Word формата «docx» использовалась библиотека Apache POI. Для конвертации файла Microsoft Word в PDF используется библиотека IcePDF. В качестве среды разработки применено приложение IntelliJ IDEA Ultimate.

Язык программирования Java - строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Права на торговую марку принадлежат корпорации Oracle. Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, для которой существует реализация виртуальной Java-машины. Java 17 – версия языка от 2021 года с длительной поддержкой.

AWT была первой попыткой Sun создать графический интерфейс для Java. Они сделали прослойку на Java, которая вызывает методы из библиотек, написанных на языке Си. Библиотечные методы создают и используют графические компоненты операционной среды. Чтобы обеспечить мультиплатформенность, пришлось унифицировать интерфейсы вызовов компонентов, из-за чего их функциональность получилась немного урезанной. Набор компонентов довольно небольшой. К примеру, в AWT нет таблиц, а в кнопках не поддерживается отображение иконок.

Вслед за AWT Sun разработала набор графических компонентов под названием Swing. Компоненты Swing полностью написаны на Java. Для отрисовки используется 2D, что принесло с собой сразу несколько преимуществ. Набор стандартных компонентов значительно превосходит AWT по разнообразию и функциональности. Стала возможной поддержка различных стилей и тем. Вместе с тем скорость работы первых версий Swing оставляла желать лучшего. Тем не менее благодаря простоте использования, богатой документации и гибкости компонентов Swing стал, пожалуй, самым популярным графическим фреймворком в Java.

FlatLaf – расширение для библиотеки Swing, созданное FormDev software и распространяемое по лицензии Apache 2.0. Оно представляет из себя тему для компонентов Swing. SoftSmithy Utility Library Swing - расширение для библиотеки Swing, созданное Florian Brunner и распространяемое по лицензии CDDL 1.0. Оно представляет из себя набор компонентов с расширенным функционалом и набор утилитарных классов.

Apache POI - это популярный API, который позволяет программистам создавать, изменять и отображать файлы MS-Office с помощью программ Java. Это библиотека с открытым исходным кодом, разработанная и распространяемая Apache Software Foundation для разработки или изменения файлов MS-Office с помощью языка Java. Apache POI содержит классы и методы для работы со всеми составными документами OLE2 MS-Office. В данном конкретном случаи нам необходимо использовать класс библиотеки Apache POI: XWPF (XML Word Processor Format) – предназначен для чтения и записи файлов с расширением .docx MS-Word.

IntelliJ IDEA Ultimate – интегрированная среда разработки программного обеспечения для языка программирования Java. Она разработана российской компанией JetBrains. Дизайн среды ориентирован на продуктивность работы программистов, позволяя сконцентрироваться на функциональных задачах, в то время как IntelliJ IDEA автоматизирует выполнение рутинных операций, таких как статический анализ кода и рефакторинг.

## 4.2 Схема программного обеспечения

Унифицированный язык моделирования (UML) – это язык для специфицирования, визуализации, конструирования и документирования программных систем, а также бизнес-моделей и прочих не программных систем. UML представляет собой объединение инженерных приемов, которые ранее успешно использовались при моделировании больших и сложных систем.

При решении сложных задач заблаговременное планирование и моделирование значительно упрощают программирование. Кроме того, вносить изменения в диаграммы классов легче, чем в исходный код.

Таким образом, первым шагом к реализации курсового проекта были построены UML диаграммы классов. Проект разделен на два логических сегмента – core и gui, отвечающих соответственно за обработку данных и за взаимодействие с пользователем. Следовательно, UML диаграмм классов две. Диаграммы классов представлены на рисунках 4.1, 4.2.

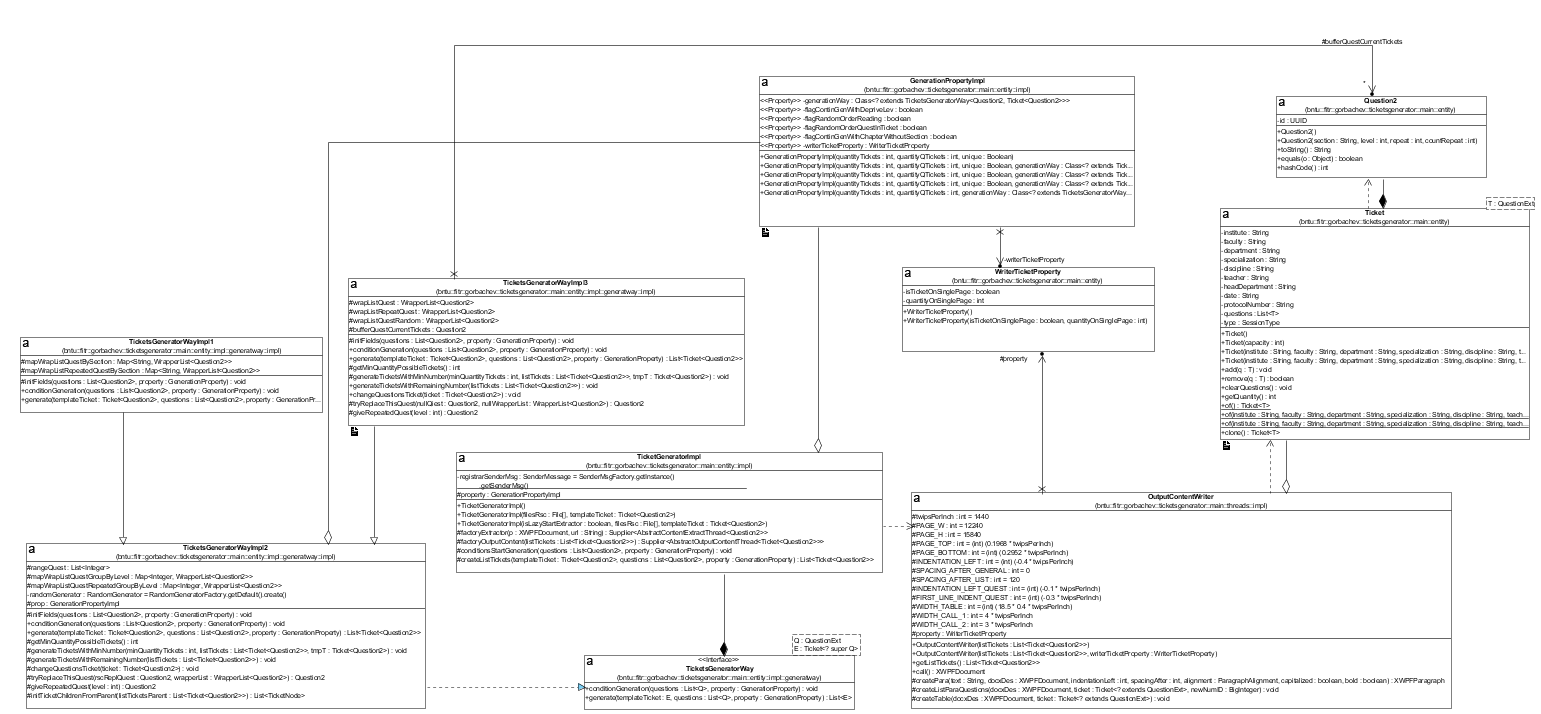


Рисунок 4.1 – Диаграмма классов бизнес модели

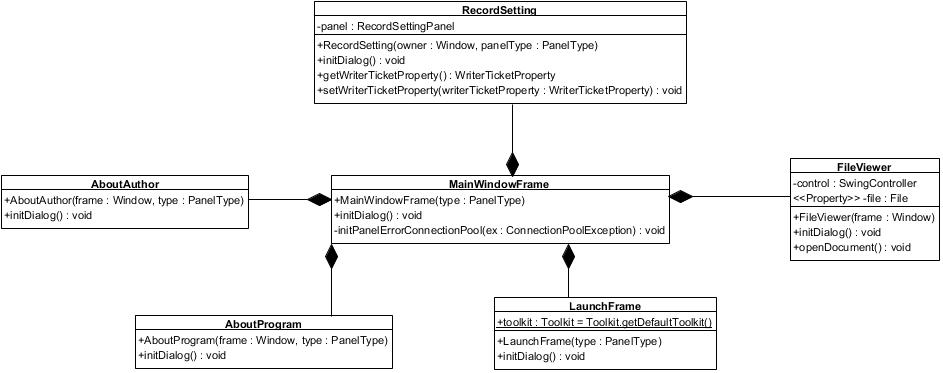
****

Рисунок 4.2 – Диаграмма классов компонента обработки отображений.

## 4.3 Описание основных классов программы

Непосредственная логика обработки данных и создания различных решений согласно приведенным в предыдущей главе алгоритмам заключена в следующих класса:

* *TicketGeneratorImpl.java*
* *GenerationPropertyImpl.java;*
* *Question2.java;*
* *Ticket.java*
* *ContentExtractor.java*
* *OutputContentWriter.java*
* *AbstractGeneratorWayImpl1.java;*
* *WriterTicketProperty.java;*

***TicketGeneratorImpl.java***

Данный класс является сущностью, определяющий методы для генерации билетов и записи сгенерированного файла.

Данный класс содержит следующие поля:

* *futureTaskExtractContent* – поток для извлечения данных из файлов fileRsc;
* *filesRsc* – список файлов ресурсов, содержащие список вопросов;
* *templateTicket* – шаблон билета, где содержится информация для его заполнения;
* *docxDec* – результирующий файл;
* *listTicket* – список билетов;

Основные методы данного класса:

* *startGenerate* – метод генерирующий файл docx, содержащий билеты;
* *writeOutputFile* – метод записывает содержимое в файл;
* *getters/setters* – для полей класса;

***Question2.java***

Данный класс представляет собой программное представление вопроса.

Данный класс содержит следующие поля:

* *id* – уникальный индификатор вопроса;
* *section* – строка для ввода темы вопроса;
* *level* – число, характеризует сложность вопроса;
* *listParagraphs* – список абзацев, является текстовым и графическим содержимым.

Основные методы данного класса:

* *getters/setters* – для полей класса;
* *clone –* для клонирования объекта;
* *add/addAll –* добавляет абзац/абзацы в вопрос;
* *remove –* удаляет абзац из вопроса;

***Ticket.java***

Класс сущности, описывающий основные характеристики билета.

Данный класс содержит следующие поля:

* *institute* – учреждение образование.
* *faculty* – факультет.
* *department* – кафедра.
* *specialization* – специальность.
* *discipline* – дисциплина.
* *teacher* – имя преподавателя.
* *headDepartment* – имя заведующего кафедрой.
* *type* – тип сессии.
* *date* – дата утверждения билета.
* *protocolNumber* – номер протокола.
* *questions* – список вопросов.

Основные методы данного класса:

* *getters/setters* – для полей класса.

***GenerationPropertyImpl.java***

Данный класс представляет собой объект передачи данных между пользовательским интерфейсом и ядром программы. Через него пользователь может настраивать различное поведение работы программы.

Данный класс содержит следующие поля:

* *quentityTickets* – указывает количество генерируемых билетов;
* *quentityQTickets* – указывает количество вопросов в билете;
* *unique* – флаг, могут ли вопросы повторятся;
* *flagRandomOrderReading*– флаг, вопросы читать произвольно.
* *flagRandomOrderQuestInTicket* – флаг, вопросы записывать произвольно.
* *flagContinGenWithgeriveLev* – флаг, позволяющий продолжить процесс генерации.
* *flagContinGenWithChapterWithoutSection* – флаг, позволяющий продолжить процесс генерации.
* *writerTicketPropety* – объект передачи данных между процессом генерации и процессом записи данных в файл.
* *generationWay* – указывает какой способ генерации применить.

Основные методы данного класса:

* *getters/setters* – для полей класса;

***WriterTicketProperty.java***

Данный класс представляет собой объект передачи данных между пользовательским графическим интерфейсом, процессом генерации билетов я процессом записи данных в файл.

Данный класс содержит следующие поля:

* *isTicketonSinglePage –* флаг, определяющий количественное расположение билетов на листе;
* *quentityOnSinglePage* – указывает количество билетов на листе.
* *getters/setters* – для полей класса;

***ContentExtractor.java***

Данный класс является потоком, цель которого – извлечь содержимое из файла .docx, предоставляющего объект класса XWPFDocument, и в результате возвращает список вопросов.

Данный класс содержит следующие поля:

* *docxFile* – объект XWPFDocument, который возвращает этот поток;
* *supplierQuestion* – фабрика по созданию вопроса;
* *urldocxFile* – строка пути считываемого файла;

Основные методы данного класса:

* *getters/setters* – для полей класса;

***OutputContentWriter.java***

Класс представляет собой поток, цель которого – записать список билетов в объект класса XWPFDocument.

Данный класс содержит следующие поля:

* listTickets – список билетов;

Основные методы данного класса:

* *getters/setters* – для полей класса;

***AbstractGeneratorWay.java***

Данные классы являются сервисами по преобразованию списка вопросов в готовый список билетов. Класс, представляет алгоритм выборку вопросов по сложности. Классы реализующие этот класс: *TicketGeneratorWay1.java, TicketGeneratorWay2.java, TicketGeneratorWay3.java.*

Данный класс содержит следующие поля:

* *prop* – объект класса *GenerationProperty.java*;
* *rangeQuest –* список последовательных чисел;
* *mapWrapListQuestGroupByLevel –* группировка списков по сложности;
* *mapWrapListQuestRepeatByLevel –* группировка списков, вопросы которых могут повторятся по сложности;

Основные методы данного класса:

* *conditionGeneration* – метод проверяет все условия генерации;
* *generate* – запускает генерацию билетов;

## 4.4 Тестовый пример и руководство пользователя

Проект представлен блок-схемой или, как ее по-другому называют, бизнес-логикой переключения окон проекта, состоящая из 8 окон взаимосвязанных между собой: Splash Screen (стартовое окно приложения), Main Window (основное окно для работы с приложением, где происходит основное взаимодействие), About Program (о программе) и About Author (об авторе) и File Viewer (окно просмотра сгенерированного файла билетов), Load File (окно загрузки файлов), Save File (окно сохранение файла), RecordSetting (окно задания свойств записи документа). Блок-схема представлена на рисунке 4.1:

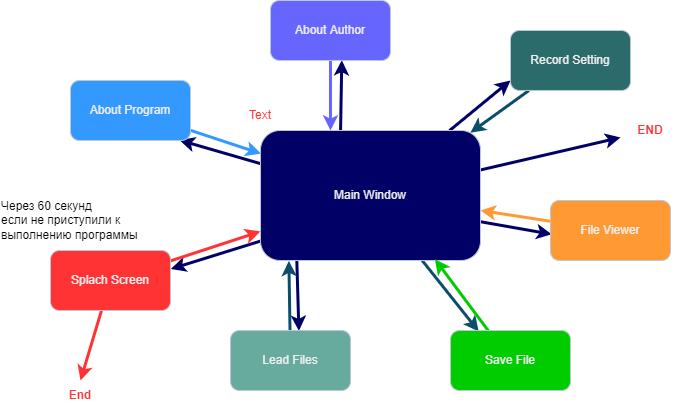
****

Рисунок 4.1 – Блок-схема бизнес-логики приложения

При запуске приложения появляется *Splash* *Screen* окно, которое закроет приложение в случаи бездействия 60 с. Данное окно приведено на рисунке 4.2.

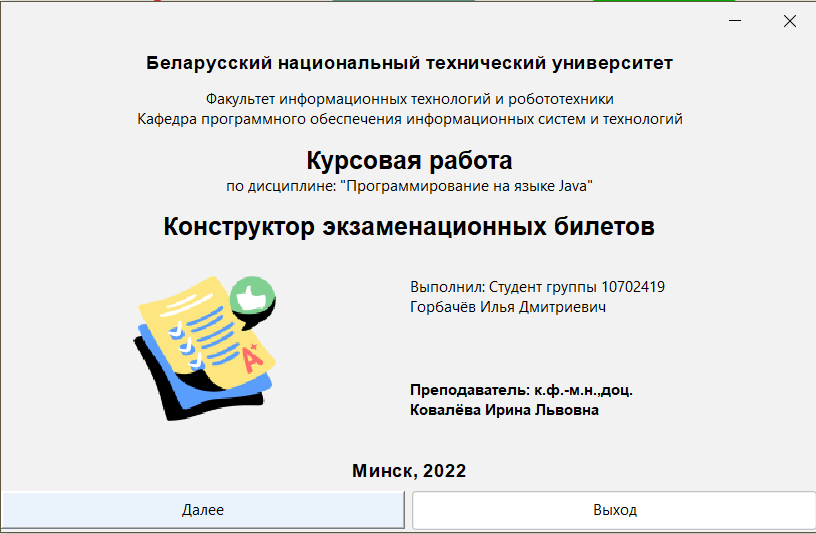


Рисунок 4.2 – Окно *Splash Screen*

При нажатии на кнопку «Выход» программа завершает свою работу. При нажатии на кнопку «Далее», появится ползунок загрузки приложения*.* Скриншот окна заставки при нажатии кнопки «Далее» приведено на рисунке 4.3.

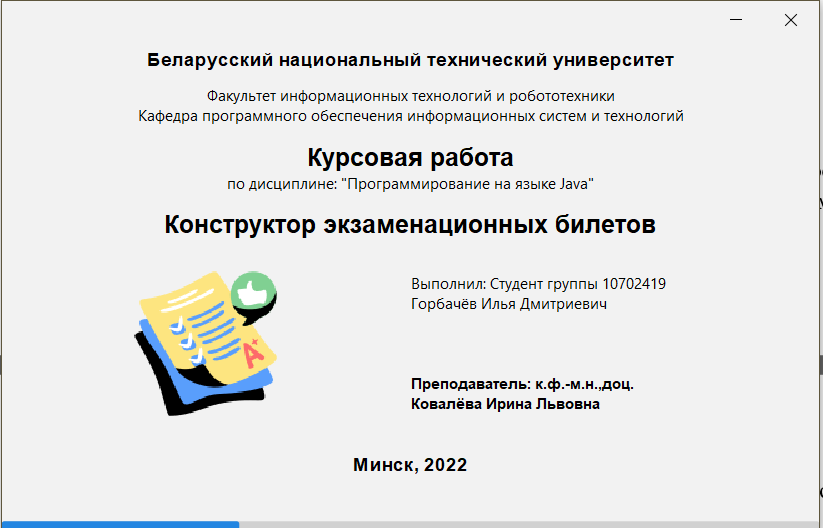


Рисунок 4.3 – Процесс загрузки приложения

При успешной загрузки приложения, будет отображено окно *Main Window –* основное окно приложения, через которое будет производится взаимодействие с пользователем.

Основное окно приложения состоит из двух секторов – левого и правого. Левый сектор – область ввода данных – предназначен для ввода исходных данных для шаблона-билета. Правый сектор – область ресурсов – предназначен для загрузки файлов и конфигураций генерации билетов*.* Скриншот окна приведено на рисунке 4.4.

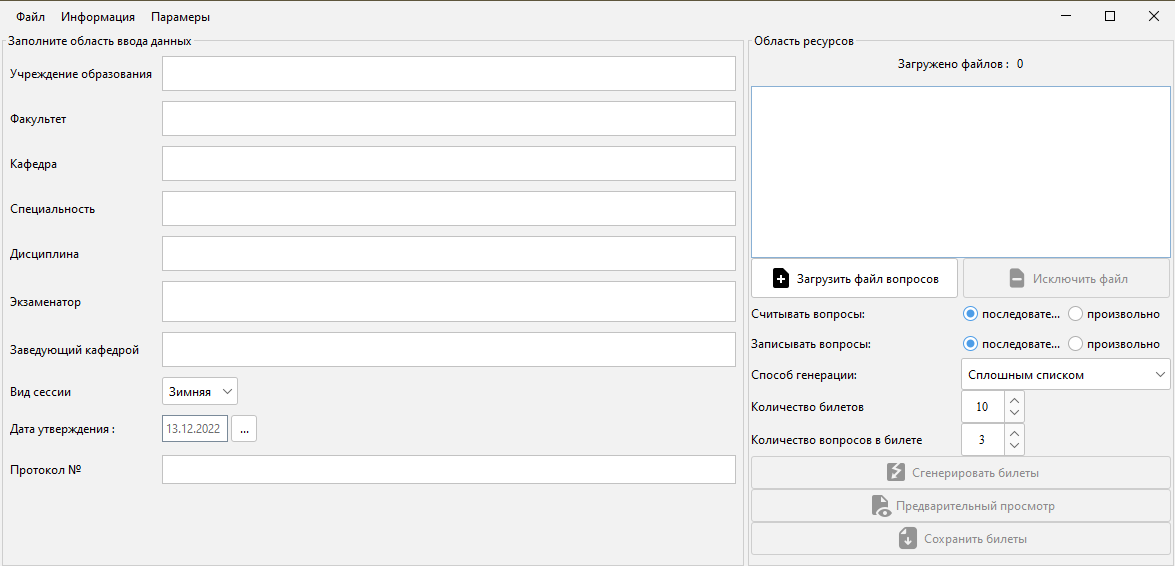


Рисунок 4.4 – Основное окно приложения

Для полей ввода соблюдается следующее правило ввода данных:

* Поле ввода *«Учреждение образования», «Факультет», «Кафедра», «Специальность», «Дисциплина»,* поддерживает символы: A-Я,А-Z, а-я, a-z, 0-9, -, \_, «, », ", }, {, ),(;
* Поле ввода *«Экзаменатор», «Заведующий кафедры»,* поддерживает любой из форматов ввода: **Фамилия Имя Отчество** или **Фамилия И О** или **Фамилия И. О.** Регистр букв важен.
* Поле ввода «Протокол №»: поддерживает любой из форматов вода: **любое целое число**, числа 5.1 4.1.13.,34.01.0.0.234.2.39113.

В правом секторе активна лишь одна кнопка: «Загрузить файл вопросов» при нажатии на которую или *ToolBar -> Файл -> Загрузить,* отображается «Файловый менеджер» по загрузке файлов Microsoft Word формата .**docx.** Как упоминалось ранее, документ Microsoft Word должен иметь заданную структуру оформления. Правило оформлению документа Microsoft Word смотрите *ПРИЛОЖЕНИЕ B.*

В данном случаи необходимо обратить внимание на атрибуты при оформление документа и на желаемый Вами способ генерации билетов, так как каждый способ генерации требует наличие своего атрибута и в случае отсутствия такого атрибута в начальном теге списка вопросов – этот список будет игнорирован, что повлечёт к другим вариантом обработок такого исключений.

В данном случаи файл оформлен под любой способ генерации и количеству вопросов в билете равна 3-ем, рисунок 4.6. Загрузим в приложение заготовленный файл нажав на кнопку «Загрузить». В результате этого выполнения в правом секторе мы увидим список загруженных файлов, а копка «Сгенерировать билеты» станет активна. Здесь же можно задать *порядок считывания/записи вопросов, способ генерации, указать количество генерируемых билетов и количество вопросов в билете.*

**

Рисунок 4.6 – Список вопросов с требуемым оформлением (цвет лишь для упрощения визуально восприятия)

При нажав на кнопку «Сгенерировать билеты» происходит отображения сообщения об процесс генерации, в процессе которого все кнопки на панели главного окна станут не активны. При необходимости процесс генерации можно отменить, нажав на кнопку «Отмена» в окне процесса-генерации, рисунок 4.6.

При несоблюдении требований к оформлению файла, программа сообщит об фатальной ошибке и укажет причину завершения процесса; необходимые кнопки вернутся в активное состояние. В случаи несоблюдения начальных *условий алгоритма,* описанных выше в разделе 3, программа сообщит об фатальной ошибке, что завершит процесс генерации, либо предложит альтернативное решение возникшего исключения, соглашаясь на который процесс генерации билетов будет восстановлен, иначе процесс завершится.

В случаи спеха генерации билетов, программа сообщит об успешности выполнения процесса. Кнопка «Предпросмотр» и «Сохранить файл» станут активны. Скриншот окна приведено на рисунке 4.7.

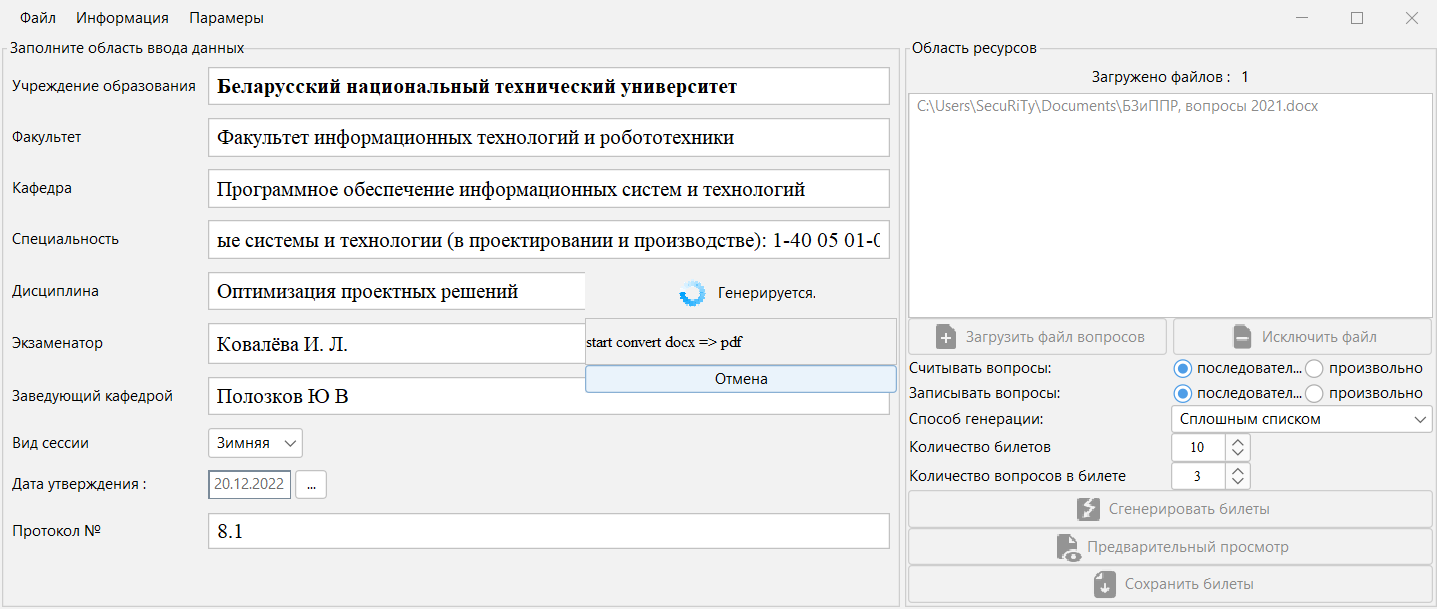


Рисунок 4.6 – Сообщение об состоянии генерации билетов

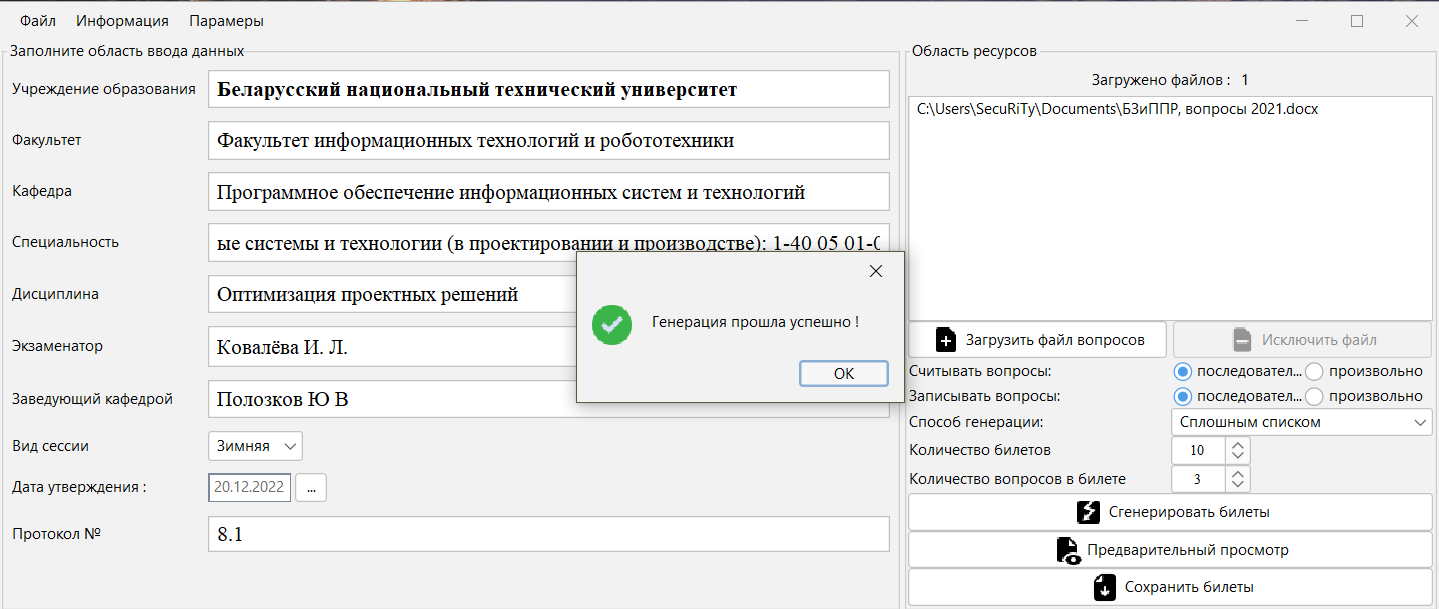


Рисунок 4.7 – Сообщение об успешной генерации билетов

Нажимая на кнопку «Предварительный просмотр» открывается диалоговое окно, отображающая сгенерированный документ. Скриншот окна приведено на рисунке 4.8.

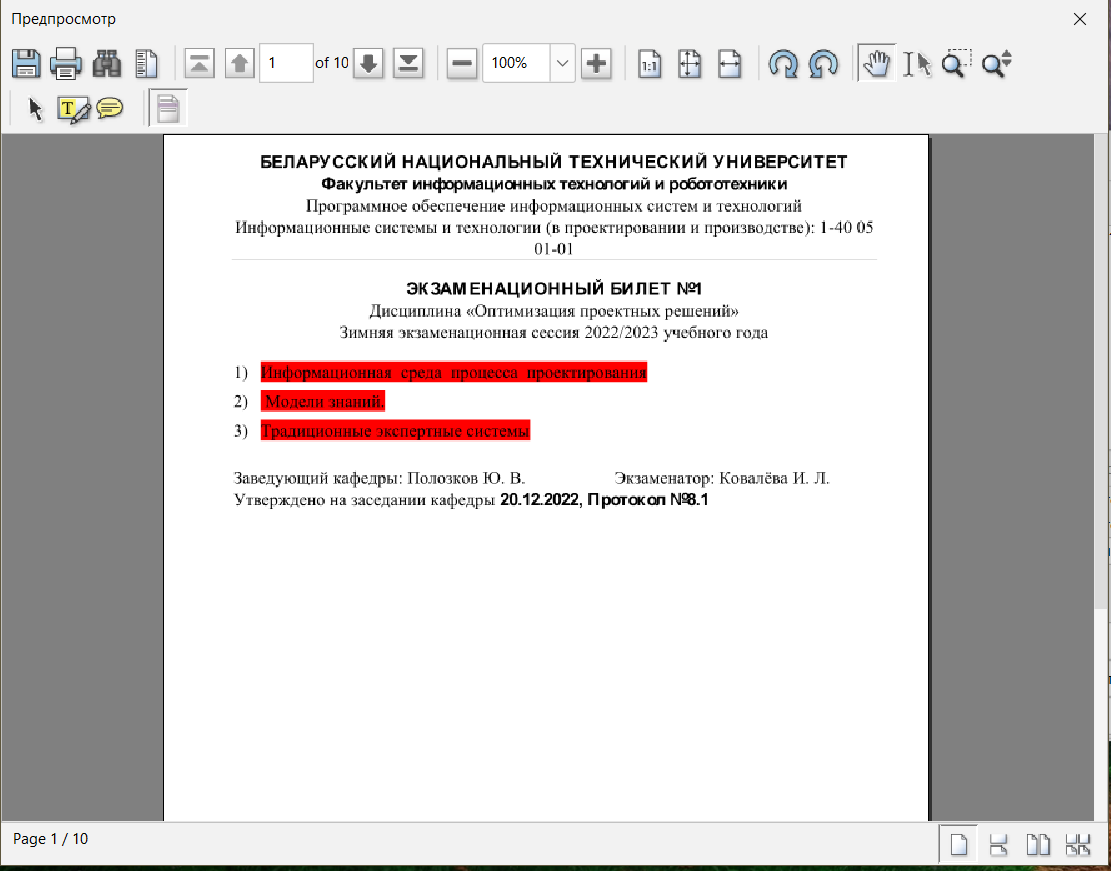


Рисунок 4.8 - Окно предварительного просмотра

Далее можно сохранить файл на диск, нажав на кнопку «Сохранить файл» либо в пути *ToolBar-> Файл -> Сохранить,* при нажатии которой появиться окно файлового менеджера по сохранению файла. После успешном сохранения файла на диск, программа предложит открыть этот документ непосредственно через внешнею программу: Microsoft Word. Скриншоты окон приведены на рисунке 4.9 – 4.10.

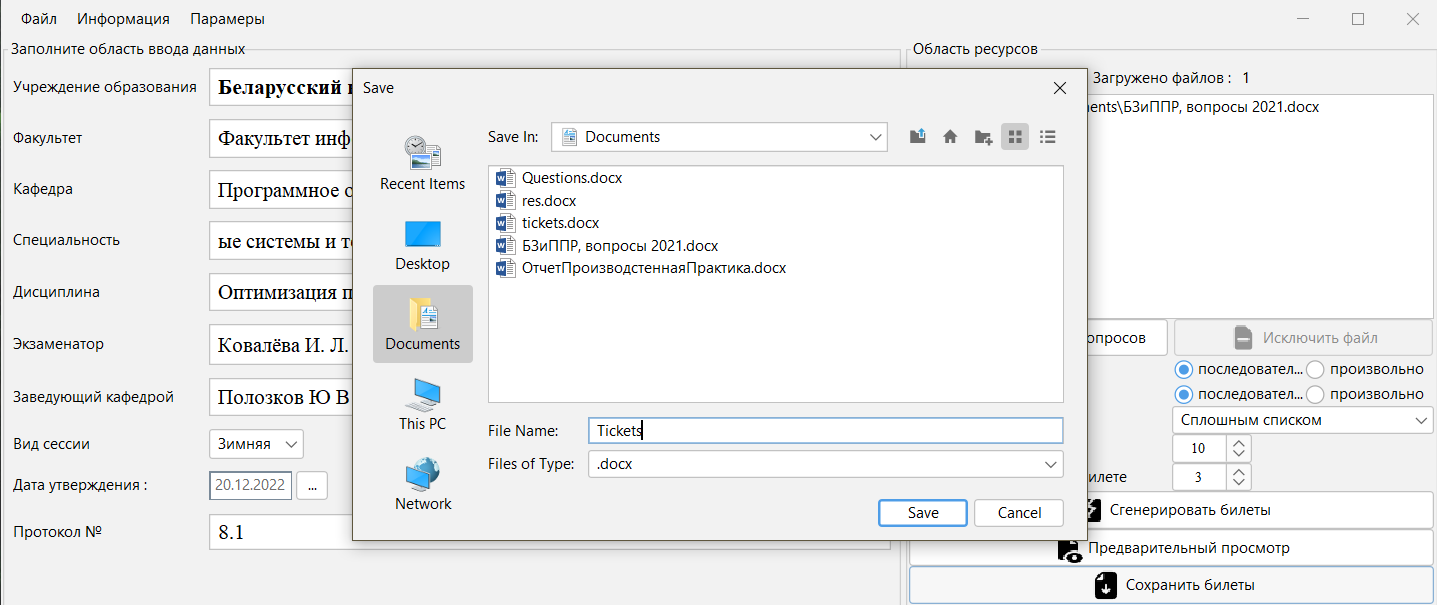
****

Рисунок 4.9 – окно файлового менеджера по сохранению сгенерированного файла

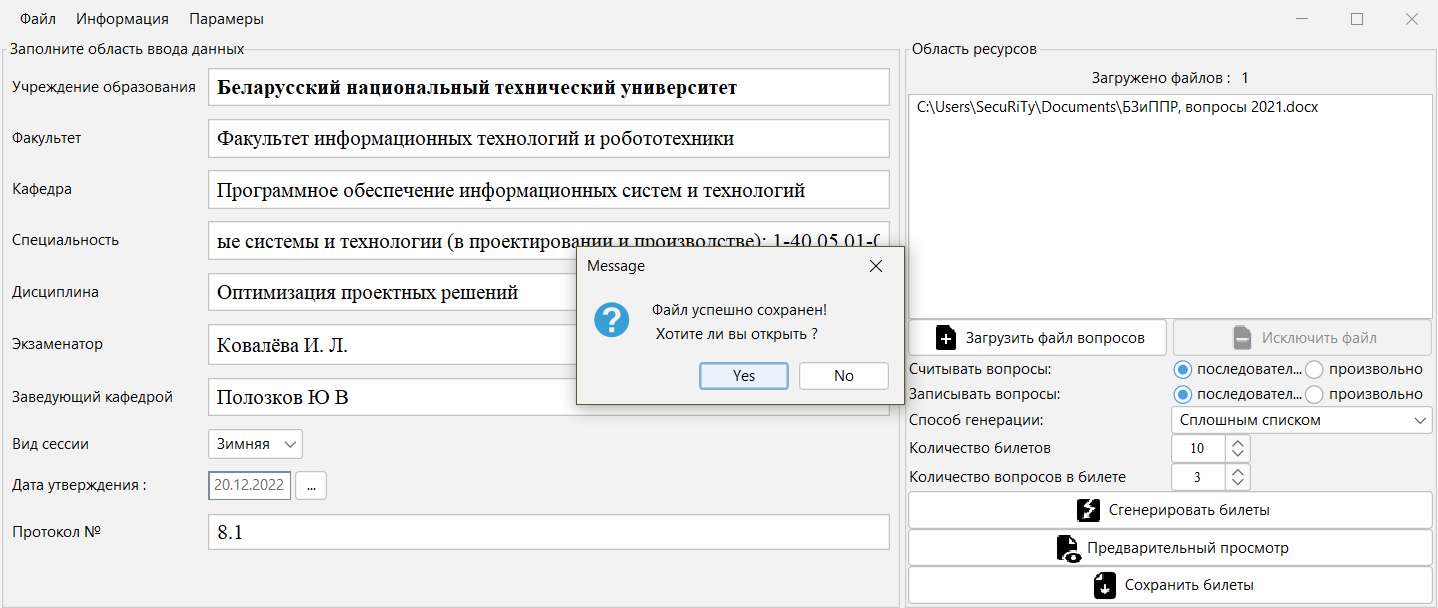


Рисунок 4.10 – Сообщение об успешном сохранении файла

Таким образом процесс формирования билетов завершен.

При необходимости размещения двух и более билетов на одном листе А4, в программе предусмотрено возможность настроить этот параметр посредством окна *Record Setting*, который можно открыть через *ToolBar ->* *Параметры* -> *Запись билетов*. Скриншот окна приведено на рисунке 4.11.

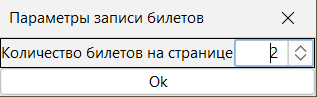


Рисунок 4.11 – Окно «Параметры записи билетов»

Результаты сгенерированных билетов тремя различными способами формирования, при последовательном извлечении/записи вопросов, представлены на рисунках 4.12 – 4.14.

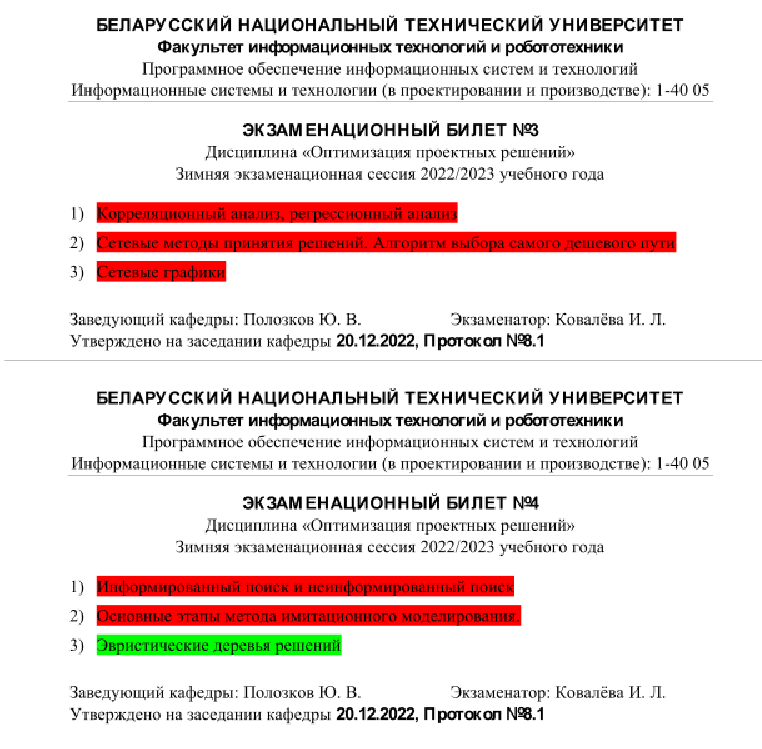


Рисунок 4.12 – Сгенерированные билеты способом «Сплошного списка»



Рисунок 4.13 - Сгенерированные билеты способом «Группировки по темам»

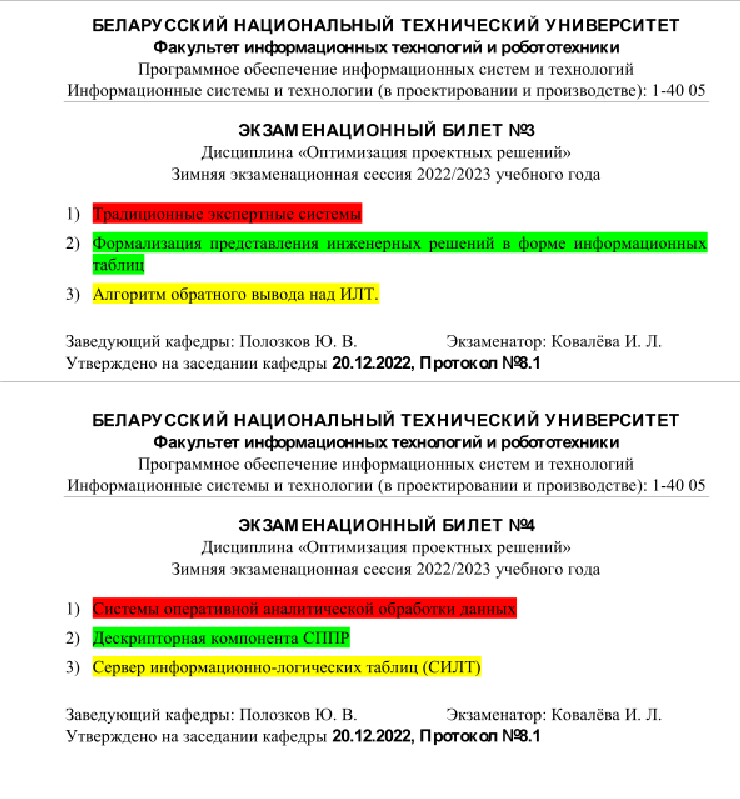
****

Рисунок 4.14 - Сгенерированные билеты способом «Группировки по сложности»

ЗАКЛЮЧЕНИЕ

В результате выполнения данного курсового проекта было создано Desktop приложение, позволяющая за короткий промежуток времени сгенерировать экзаменационный бланк билетов, облегчив важнейшую частью учебно-методической деятельности преподавателя любого учебного заведения.

Данное приложение предоставляет достаточную гибкость в выборе метода составления билетов, позволяющего распределить вопросы как в сплошном порядке, так и сгруппировать вопросы по сложности или темам; позволяет преподавателю иметь широкий спектр возможностей по форматированию содержимого вопроса и управлять их повторяемостью, а также настройки параметр размещения двух и более билетов на одном листе А4. На данный момент сформирован дистрибутив для *ОС Windows*.

Применение данного приложения возможно в любом учебном заведении страны Республики Беларусь, а также стран со схожей структурой обучения.

Дальнейшее развитие приложения может заключаться в расширении бизнес-функционала и расширение графического функционала пользователя. Так, для устранения необходимости вручную вводить данные шаблона билета, предоставить пользователю сохранять ранее веденные данные в БД встроенной локально в это приложение. Предоставить графический интерфейс, позволяющий пользователю напрямую в режиме runtimeизменять значение атрибутов вопроса, в случаи возникновения различных альтернативных ситуаций, которые возникают в процесс генерирования экзаменационных билетов, устраняя необходимость вносить правки в файле-источнике, тем самым еще более ускорить процесс достижения конечного результата. Возможно расширение списка поддерживаемых форматов файлов для сохранения результата. Так как язык реализации поддерживает кроссплатформенность, полезным нововведением будет создание дистрибутивов приложения для ОС Linux, Mac OS и прочих операционных систем

**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. Экзаменационные билеты. Составление экзаменационных билетов: [сайт]. [2015-2022]. URL: <https://mydocx.ru/7-22659.html?ysclid=lber276ism32534886> (дата обращения: 08.12.2022).
2. Как составить экзаменационные билеты. Автор «КакПросто!» [сайт]. [от 16.09.2014]. URL: <https://www.kakprosto.ru/kak-903276-kak-sostavlyat-ekzamenacionnye-bilety?ysclid=lberd83ngt224375202> (дата обращения: 08.12.2022).
3. Программа "Генератор билетов" (Ticket Generator) [Электронный ресурс]. URL: <https://kaiu.narod.ru/Genbilet/Genbilet.html> (дата обращения: 04.10.2022)
4. Создание билетов для зачета или экзамена по спискам вопросов различных типов. PLANETCALC: [Электронный ресурс]. [2021].URL: <https://planetcalc.ru/9579/> (дата обращения: 16.10.2022).
5. Генератор экзаменационных билетов. Голубев А, Кулев М. [Электронный ресурс]. [2021]. URL: <https://issuu.com/galex007/docs/icmc-2005-examcardgenerator_rus> (дата обращения: 22.09.2022)

# ПРИЛОЖЕНИЕА*.* ИСХОДНЫЙКОДПРИЛОЖЕНИЯ

**package** bntu.fitr.gorbachev.ticketsgenerator.main.entity;

**public abstract class** AbstractTicketGenerator<Q **extends** QuestionExt, T **extends** Ticket<Q>>  
 **implements** Callable<List<Q>> {  
 **private** Future<List<Q>> **futureTaskExtractContent**;  
 **protected** File[] **filesRsc**;  
 **private** XWPFDocument **docxDec**;  
 **protected** T **templateTicket**;  
 **protected** List<T> **listTicket**;  
GenerationProperty **generationProperty**  
 **public** AbstractTicketGenerator() {  
 }  
**public** AbstractTicketGenerator(File[] filesRsc, T templateTicket) {  
 **if** (filesRsc == **null** || templateTicket == **null**) {  
 **throw new** NullPointerException(**"Initialization attributes is needed condition"**);  
 }  
 **this**.**filesRsc** = filesRsc;  
 **this**.**templateTicket** = templateTicket;  
 *// by default run start extracting threads* **this**.runStartExtractorThreads();  
 }  
**public** AbstractTicketGenerator(**boolean** isLazyStartExtractor, File[] filesRsc, T templateTicket) {  
 **if** (filesRsc == **null** || templateTicket == **null**) {  
 **throw new** NullPointerException(**"Initialization attributes is needed condition"**);  
 }  
 **this**.**filesRsc** = filesRsc;  
 **this**.**templateTicket** = templateTicket;  
 *// run start extraction threadS if is false, else* **if** (!isLazyStartExtractor) **this**.runStartExtractorThreads();  
 }**public** XWPFDocument getDocxDec() {  
 **return docxDec**;  
 }

**public** List<T> getListTicket() {  
 **return listTicket**;  
 }  
**public** AbstractTicketGenerator<Q, T> setConstParam(File[] filesRsc, T templateTicket) {  
 **if** (filesRsc == **null** || templateTicket == **null**) {  
 **throw new** NullPointerException(**"Initialization attributes is needed condition"**);  
 }  
 **this**.**templateTicket** = templateTicket;  
 **this**.**filesRsc** = filesRsc;  
 **return this**;  
 }  
**public** List<Q> getListQuestions() **throws** ExecutionException, InterruptedException {  
 **if** (**futureTaskExtractContent** == **null**) { *// this notifies will be is as reason futureTask == null* **throw new** ExecutionException(**"Apparently, the ticket generator was been created with parameter:"** +  
 **"isLazyStartExtractor == true"**,  
 **new** NullPointerException(**"futureTaskExtractorContent == null"**));  
 }  
 **return futureTaskExtractContent**.get();  
 }**public void** writeOutputFile(File fileDes) **throws** IOException {  
 **if** (**docxDec** == **null**) **return**;  
 **try** (FileOutputStream outputThread = **new** FileOutputStream(fileDes)) {  
 **docxDec**.write(outputThread);  
 }  
 }  
**private void** runStartExtractorThreads() {  
 ExecutorService executorService = Executors.*newSingleThreadExecutor*();  
 **this**.**futureTaskExtractContent** = executorService.submit(**this**);  
 executorService.shutdown();  
 }

@Override  
 **public** List<Q> call() **throws** Exception {  
 { List<Q> generalList = **new** ArrayList<>();  
 FileInputStream[] inputTreads = **new** FileInputStream[**filesRsc**.**length**];  
 XWPFDocument[] docxRsc = **new** XWPFDocument[**filesRsc**.**length**];  
 ExecutorService executor = Executors.*newFixedThreadPool*(**filesRsc**.**length**);  
 List<Future<List<Q>>> futures = **new** ArrayList<>(**filesRsc**.**length**);  
 **try** {  
 **for** (**int** i = 0; i < inputTreads.**length**; ++i) {  
 inputTreads[i] = **new** FileInputStream(**filesRsc**[i]);  
 docxRsc[i] = **new** XWPFDocument(inputTreads[i]);  
 *// thread launch* AbstractContentExtractThread<Q> extractor = factoryExtractor(docxRsc[i], **filesRsc**[i].getName())  
 .get();  
 futures.add(executor.submit(extractor)); }  
 **for** (Future<List<Q>> futureTask : futures) {  
 generalList.addAll(futureTask.get()); }  
 } **catch** (ExecutionException e) {  
 Throwable cause = e.getCause();  
 System.***out***.println(cause.getMessage());  
 **throw new** ExecutionException(cause.getMessage(), **null**);  
 } **catch** (InterruptedException ignored) {  
 } **finally** {  
 *// closing pool bntu.fitr.gorbachev.ticketsgenerator.main.threads and cancel all executing tasks and tasks in the queue* executor.shutdownNow();  
 **for** (**var** thread : inputTreads) { *// closing files* **if** (thread != **null**) {  
 thread.close();  
 }  
 }  
 }  
 **return** generalList;  
 }  
 }  
**public final void** startGenerate(GenerationProperty generationProperty)  
 **throws** GenerationConditionException, ExecutionException, InterruptedException {  
  
 **this**.**generationProperty** = generationProperty;  
 checkedNecessarilyConditions();  
  
 *// run staring thread for extract content from docx file* **if** (**futureTaskExtractContent** == **null**) **this**.runStartExtractorThreads();  
  
 List<Q> listQuestions;  
 **try** {  
 listQuestions = **futureTaskExtractContent**.get(); *// await answer* } **catch** (InterruptedException e) { *// in case interrupted thread* **futureTaskExtractContent**.cancel(**true**); *// then also interrupt extract-thread* **throw new** InterruptedException(e.getMessage()); *// throw this exception one level higher* }  
  
 conditionsStartGeneration(listQuestions, **this**.**generationProperty**);  
  
 **listTicket** = createListTickets(**templateTicket**, listQuestions, **this**.**generationProperty**);  
  
 *// lunch output content formation thread* AbstractOutputContentThread<T> threadWriteTickets = factoryOutputContent(**listTicket**).get();  
 ExecutorService executorService = Executors.*newSingleThreadExecutor*();  
 Future<XWPFDocument> futureTaskOutputContent = executorService.submit(threadWriteTickets);  
 executorService.shutdown();  
 **try** {  
 **docxDec** = futureTaskOutputContent.get(); *// await answer* } **catch** (InterruptedException e) {  
 futureTaskOutputContent.cancel(**true**);  
 **throw new** InterruptedException(e.getMessage()); *// throw this exception one level higher* } }  
**protected abstract** Supplier<AbstractContentExtractThread<Q>> factoryExtractor(XWPFDocument p, String url);  
**protected abstract** List<T> createListTickets(T templateTicket, List<Q> questions, GenerationProperty property);**protected abstract** Supplier<AbstractOutputContentThread<T>> factoryOutputContent(List<T> listTickets);  
**private void** checkedNecessarilyConditions() **throws** GenerationConditionException {  
 *// Throw Exception if incorrect entered parameters method* **if** (**generationProperty** == **null**) {  
 **throw new** GenerationConditionException(**new** NullPointerException(**"Your need initialize generation property"**));  
 } **else if** (**generationProperty**.getQuantityTickets() <= 0) {  
 **throw new** GenerationConditionException(**"Incorrect quality entered tickets"**);  
 } **else if** (**generationProperty**.getQuantityQTickets() <= 0) {  
 **throw new** GenerationConditionException(**"insufficient number of questions to ensure "** +  
 **"\nthat questions are not repeated in stupid tickets."**);  
 } **else if** (**filesRsc** == **null** || **templateTicket** == **null**) {  
 **throw new** GenerationConditionException(**"Was invoked constructor without parameters."** +  
 **"You need to initialize attributes: filesRsc, templateTicket "** +  
 **"through methods setter"**); } }**protected abstract void** conditionsStartGeneration(List<Q> questions, GenerationProperty generationProperty)  
 **throws** GenerationConditionException;  
}**package** bntu.fitr.gorbachev.ticketsgenerator.main.entity.impl;  
**public class** TicketGeneratorImpl **extends** AbstractTicketGenerator<Question2, Ticket<Question2>> {  
 **protected** GenerationPropertyImpl **property**;  
 **private final static** SenderMessage ***registrarSenderMsg*** = SenderMsgFactory.*getInstance*()  
 .getSenderMsg();  
 **public** TicketGeneratorImpl() {  
 }  
**public** TicketGeneratorImpl(File[] filesRc, Ticket<Question2> templateTicket) {  
 **super**(filesRsc, templateTicket);  
 } **public** TicketGeneratorImpl(**boolean** isLazyStartExtractor, File[] filesRsc, Ticket<Question2> templateTicket) {  
 **super**(isLazyStartExtractor, filesRsc, templateTicket);  
 } @Override  
 **protected** Supplier<AbstractContentExtractThread<Question2>> factoryExtractor(XWPFDocument p, String url) {  
 ***registrarSenderMsg***.sendMsg(**"extract data:"** + url);  
 **return** () -> **new** ContentExtractor(p, url);  
 }  
 @Override  
 **protected** Supplier<AbstractOutputContentThread<Ticket<Question2>>> factoryOutputContent(List<Ticket<Question2>> listTickets) {  
 ***registrarSenderMsg***.sendMsg(**"creating tickets using the docx template"**);  
 **return** () -> **new** OutputContentWriter(listTickets, **property**.getWriterTicketProperty());  
 } @Override  
 **protected void** conditionsStartGeneration(List<Question2> questions, GenerationProperty property)  
 **throws** GenerationConditionException {  
 ***registrarSenderMsg***.sendMsg(**"checking the conditions for starting generation"**);  
 GenerationPropertyImpl prop = (GenerationPropertyImpl) property;  
 **this**.**property** = prop;  
 TicketGeneratorManager.*getGenerator*(prop.getGenerationWay()).conditionGeneration(questions, property);  
 }  
 @Override  
 **protected** List<Ticket<Question2>> createListTickets(Ticket<Question2> templateTicket, List<Question2> questions,  
 GenerationProperty property) {  
 GenerationPropertyImpl prop = (GenerationPropertyImpl) property;  
 ***registrarSenderMsg***.sendMsg(**"creation list tickets"**);  
 **return** TicketGeneratorManager.*getGenerator*(prop.getGenerationWay()).generate(templateTicket, questions, property);  
 }  
}

**package** bntu.fitr.gorbachev.ticketsgenerator.main.entity;  
@Getter  
**public class** Question2 **extends** QuestionExt {  
 **private final** UUID **id**;  
 {  
 **id** = UUID.*randomUUID*();  
 }  
 **public** Question2() {  
 }  
 **public** Question2(String section, **int** level, **int** repeat, **int** countRepeat) {  
 **super**(section, level, repeat);  
 }  
 @Override  
 **public** String toString() {  
 **return** getListParagraphs().get(0).getText();  
 }  
@Override  
 **public boolean** equals(Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (o == **null** || getClass() != o.getClass()) **return false**;  
 Question2 question2 = (Question2) o;  
 **return** Objects.*equals*(**id**, question2.**id**);  
 }  
 @Override  
 **public int** hashCode() {  
 **return** Objects.*hash*(**id**);  
 }  
}

**package** bntu.fitr.gorbachev.ticketsgenerator.main.entity;@Setter  
@Getter  
@ToString(callSuper = **true**)  
**public class** Ticket<T **extends** QuestionExt> **implements** Cloneable {  
 **private** String **institute**;  
 **private** String **faculty**;  
 **private** String **department**;  
 **private** String **specialization**;  
 **private** String **discipline**;  
 **private** String **teacher**;  
 **private** String **headDepartment**;  
 **private** SessionType **type**;  
 **private** String **date**;  
 **private** String **protocolNumber**;  
 **private** List<T> **questions**; { **institute** = **""**;  
 **faculty** = **""**;  
 **department** = **""**;  
 **specialization** = **""**;  
 **discipline** = **""**;  
 **teacher** = **""**; **headDepartment** = **""**;  
 **type** = SessionType.***AUTUMN***;  
 **date** = **""**; **protocolNumber** = **""**;  
 }**public** Ticket() { **questions** = **new** ArrayList<>(4);  
 } **public** Ticket(**int** capacity) {  
 **questions** = **new** ArrayList<>(capacity);  
 }  
**public** Ticket(String institute, String faculty, String department,  
 String specialization, String discipline,  
 String teacher, String headDepartment, SessionType type,  
 String date, String protocolNumber) {  
 **this**();  
 **this**.**institute** = institute;  
 **this**.**faculty** = faculty;  
 **this**.**department** = department;  
 **this**.**specialization** = specialization;  
 **this**.**discipline** = discipline;  
 **this**.**teacher** = teacher;  
 **this**.**headDepartment** = headDepartment;  
 **this**.**type** = type;  
 **this**.**date** = date;  
 **this**.**protocolNumber** = protocolNumber;  
 }  
**public** Ticket(String institute, String faculty, String department,  
 String specialization, String discipline,  
 String teacher, String headDepartment, SessionType type,  
 String date, String protocolNumber, **int** capacity) {  
 **this**.**institute** = institute;  
 **this**.**faculty** = faculty;  
 **this**.**department** = department;  
 **this**.**specialization** = specialization;  
 **this**.**discipline** = discipline;  
 **this**.**teacher** = teacher;  
 **this**.**headDepartment** = headDepartment;  
  
 **this**.**type** = type;  
 **this**.**date** = date;  
 **this**.**protocolNumber** = protocolNumber;  
 **questions** = **new** ArrayList<>(capacity);  
 }**public void** add(T q) { **questions**.add(q); }  
**public boolean** remove(T q) { **return questions**.remove(q);  
 }**public void** clearQuestions() { **questions**.clear();  
 **public int** getQuantity() { **return questions**.size();  
 } **public static** <T **extends** QuestionExt> Ticket<T> of() {  
 **return new** Ticket<>(); }  
 **public static** <T **extends** QuestionExt> Ticket<T> of(String institute, String faculty, String department,  
 String specialization, String discipline,  
 String teacher, String headDepartment, SessionType type,  
 String date, String protocolNumber) {  
 **return new** Ticket<>(institute, faculty, department,  
 specialization, discipline, teacher, headDepartment, type,  
 date, protocolNumber);  
 }  
 **public static** <T **extends** QuestionExt> Ticket<T> of(String institute, String faculty, String department,  
 String specialization, String discipline,  
 String teacher, String headDepartment, SessionType type,  
 String date, String protocolNumber, **int** cap) {  
 **return new** Ticket<>(institute, faculty, department,  
 specialization, discipline, teacher, headDepartment, type,  
 date, protocolNumber, cap);  
 }  
**public enum** SessionType {  
 ***AUTUMN***(**"Осенняя"**), ***WINTER***(**"Зимняя"**),  
 ***SPRING***(**"Весенняя"**), ***SUMMER***(**"Летняя"**),  
 ***NON***(**"не определен"**);  
 **private final** String **name**;  
SessionType(String name) {  
 **this**.**name** = name;  
 }  
@Override  
 **public** String toString() {  
 **return name**;  
 }  
 }  
@Override  
 **public** Ticket<T> clone() {  
 Ticket<T> clone = **null**;  
 **try** {  
 clone = (Ticket<T>) **super**.clone();  
 clone.**questions** = **new** ArrayList<>(**this**.**questions**);  
 } **catch** (CloneNotSupportedException e) {  
 e.printStackTrace();  
 } **return** clone; }}

**package** bntu.fitr.gorbachev.ticketsgenerator.main.entity.impl.generatway.impl;  
**public class** TicketsGeneratorWayImpl2 **implements** TicketsGeneratorWay<Question2, Ticket<Question2>> {  
 **protected** GenerationPropertyImpl **prop**;  
 **protected** List<Integer> **rangeQuest**;  
 **protected** Map<Integer, WrapperList<Question2>> **mapWrapListQuestGroupByLevel**;  
 **protected** Map<Integer, WrapperList<Question2>> **mapWrapListQuestRepeatedGroupByLevel**;  
 **private final** RandomGenerator **randomGenerator** = RandomGeneratorFactory.*getDefault*().create();  
 **protected void** initFields(List<Question2> questions, GenerationProperty property) {  
 *// Initialize here, that don't duplicate extra code* **prop** = (GenerationPropertyImpl) property;  
 **rangeQuest** = IntStream.*rangeClosed*(1, **prop**.getQuantityQTickets()).boxed().collect(Collectors.*toList*());  
 ExecutorService service = Executors.*newFixedThreadPool*(2);  
 List<Future<?>> futures = **new** ArrayList<>(2);  
 futures.add(service.submit(() -> {  
 **mapWrapListQuestGroupByLevel** = questions.stream()  
 .collect(Collectors.*groupingBy*(Question2::getLevel, TreeMap::**new**,  
 Collectors.*collectingAndThen*(Collectors.*toList*(), WrapperList::*of*))) }));  
 futures.add(service.submit(() -> {  
 **mapWrapListQuestRepeatedGroupByLevel** = questions.stream()  
 .collect(Collectors.*groupingBy*(Question2::getLevel, TreeMap::**new**,  
 Collectors.*filtering*(q -> q.getRepeat() > 0,  
 Collectors.*collectingAndThen*(Collectors.*toList*(), WrapperList::*of*)))); }));  
 service.shutdown();  
 **for** (**var** future : futures) {  
 **try** {  
 future.get();  
 } **catch** (InterruptedException | ExecutionException e) {  
 e.printStackTrace(); } } } @Override  
 **public void** conditionGeneration(List<Question2> questions, GenerationProperty property)  
 **throws** GenerationConditionException {  
 initFields(questions, property);  
 Map<Boolean, List<Integer>> mapRang = **rangeQuest**.stream()  
 .collect(Collectors.*partitioningBy*(**mapWrapListQuestGroupByLevel**::containsKey));  
 List<Integer> findsNonMatchLevel = **mapWrapListQuestGroupByLevel**.keySet().stream()  
 .filter(Predicate.*not*(mapRang.get(**true**)::contains)).toList();  
 **if** (!mapRang.get(**false**).isEmpty()) {  
 System.***out***.println(mapRang.get(**true**));  
 System.***out***.println(mapRang.get(**false**));  
 *// fatal exception* **throw new** GenerationConditionException(  
 **"Вы указали количество вопросов в билете: "** + **prop**.getQuantityQTickets() + **".\n"** +  
 **"Отсутствуют вопросы со сложностью: "** + mapRang.get(**false**) + **".\n"** +  
 ((findsNonMatchLevel.isEmpty()) ? **""** : **"Однако найдены вопросы со сложностью: "** + findsNonMatchLevel + **",\n"**) +  
 **"необходимо указать сложность вопросов в приделах: [1;"** + **prop**.getQuantityQTickets() + **"]"** );  
 } **else if** (!findsNonMatchLevel.isEmpty()) {  
 **if** (!**prop**.isFlagContinGenWithDepriveLev()) {  
 *// exception allowed to continue generation. That continue generation needed set flag == true* **throw new** FindsNonMatchingLevel(**"Вы указали количество вопросов в билете: "** + **prop**.getQuantityQTickets() + **"\n"** +  
 **"Кроме вопросов со сложностью: "** + mapRang.get(**true**) + **", что позволяет сгенерировать билеты,\n"** +  
 **"были найдены вопросы сложности: "** + findsNonMatchLevel + **".\n"** +  
 **"Выборка вопросов будет производится только в пределах: [1; "** + **prop**.getQuantityQTickets() + **"]\n"** );  
 } **else** {  
 *// just remove unnecessary levels from global mapListQuest* **for** (**int** unnecessaryLevel : findsNonMatchLevel) {  
 **mapWrapListQuestGroupByLevel**.remove(unnecessaryLevel);  
 **mapWrapListQuestRepeatedGroupByLevel**.remove(unnecessaryLevel); } } }  
Set<Map.Entry<Integer, Integer>> entryQuantityNotEnough = **new** HashSet<>(**rangeQuest**.size());  
 **for** (**var** entry : **mapWrapListQuestRepeatedGroupByLevel**.entrySet()) {  
 **int** totalQuantity = entry.getValue().stream().mapToInt(Question2::getRepeat).sum();  
 totalQuantity = **prop**.getQuantityTickets() - totalQuantity - **mapWrapListQuestGroupByLevel**.get(entry.getKey()).size();  
 **if** (totalQuantity > 0) {  
 entryQuantityNotEnough.add(**new** AbstractMap.SimpleEntry<>(entry.getKey(), totalQuantity)); } }  
 **if** (**prop**.isUnique()) { *// if true, this is means then we require that USER control repeated questions  
 // then check really user take into account the conditions for generation tickets* **if** (!entryQuantityNotEnough.isEmpty()) {  
 *// exception allowed to continew generation* **throw new** NumberQuestionsRequireException(**"Вы указали: "** + **prop**.getQuantityQTickets() + **" вопросов в билете.\n"** +  
 **"Требуется, чтобы количество вопросов в каждой из сложностей суммарно\n"** +  
 **" был равен как минимум: "** + **prop**.getQuantityTickets() + **""** +  
 **" (с учётом указанного Вами количество повторения).\n"** +  
 **"Не достаточно вопросов у сложности:\n"** +  
 entryQuantityNotEnough.stream()  
 .map(e -> e.getKey() + **" => в количестве: "** + e.getValue())  
 .collect(Collectors.*joining*(**"\n"**)) + **".\n"** +  
 **"Среди вопросов, у которых указано число повторений будет\n"** +  
 **"равномерно увеличено недостающее число повторений, если таковые имеются,\n"** +  
 **"иначе вопросы будут выбраны произвольно."**);  
 }  
 } **else** { **for** (**var** entry : entryQuantityNotEnough) {  
 List<Question2> listRepeatedQuest = **mapWrapListQuestRepeatedGroupByLevel**.get(entry.getKey());  
 **if** (!listRepeatedQuest.isEmpty()) {  
 **int** fullPass = entry.getValue() / listRepeatedQuest.size();  
 **int** quantityQuestForIncrease = entry.getValue() - fullPass \* listRepeatedQuest.size();  
  
 **for** (**int** i = 0; i < listRepeatedQuest.size(); i++) {  
 Question2 repQuest = listRepeatedQuest.get(i);  
  
 **int** rep = fullPass + repQuest.getRepeat();  
 **if** (i < quantityQuestForIncrease) {  
 ++rep;  
 }  
 repQuest.setRepeat(rep); } } } } }  
 @Override  
 **public** List<Ticket<Question2>> generate(Ticket<Question2> templateTicket, List<Question2> questions,  
 GenerationProperty property) {  
 List<Ticket<Question2>> listTickets = **new** ArrayList<>(property.getQuantityTickets());  
  
 **if** (**prop**.isFlagRandomOrderReading()) {  
 **mapWrapListQuestGroupByLevel**.forEach((lev, wrapListQ) -> Collections.*shuffle*(wrapListQ.getList()));  
 **mapWrapListQuestRepeatedGroupByLevel**.forEach((lev, wrapListQ) -> Collections.*shuffle*(wrapListQ.getList())); } **int** minQuantityTickets = getMinQuantityPossibleTickets();  
 generateTicketsWithMinNumber(minQuantityTickets, listTickets, templateTicket);  
 *// checking on to continue generation additional tickets* **if** (**prop**.getQuantityTickets() == minQuantityTickets) { *// if equals, then complete generate* **return** listTickets; }  
 *// else continue generation* generateTicketsWithRemainingNumber(listTickets);  
 **return** listTickets; }  
 **protected int** getMinQuantityPossibleTickets() {  
 **int** minListSize = **mapWrapListQuestGroupByLevel**.entrySet().stream()  
 .min(Map.Entry.*comparingByValue*(Comparator.*comparingInt*(WrapperList::size)))  
 .orElseThrow().getValue().size();  
 *// generate with minimum possible and needed number tickets* **return** Integer.*min*(**prop**.getQuantityTickets(), minListSize); }  
 **protected void** generateTicketsWithMinNumber(**final int** minQuantityTickets, **final** List<Ticket<Question2>> listTickets, Ticket<Question2> tmpT) {  
 tmpT.clearQuestions();  
 **for** (**int** indexArray = 0; indexArray < minQuantityTickets; ++indexArray) {  
 Ticket<Question2> ticket = tmpT.clone();  
 **if** (**prop**.isFlagRandomOrderQuestInTicket()) Collections.*shuffle*(**rangeQuest**);  
 **for** (**var** level : **rangeQuest**) {  
 ticket.add(**mapWrapListQuestGroupByLevel**.get(level).next());  
 } listTickets.add(ticket); } }  
 **protected void** generateTicketsWithRemainingNumber(**final** List<Ticket<Question2>> listTickets) {  
 List<TicketNode> listNodes = initTicketChildrenFromParent(listTickets);  
 listNodes.stream()  
 .flatMap(parentTicketNode -> parentTicketNode.getChildrenNodes().stream())  
 .map(childTicketNode -> {  
 Ticket<Question2> ticket = childTicketNode.getTicket();  
 System.***out***.println(**"childern TIcket: "** + ticket);  
 **if** (**prop**.isFlagRandomOrderQuestInTicket()) Collections.*shuffle*(**rangeQuest**);  
 **this**.changeQuestionsTicket(ticket);  
 System.***out***.println(ticket + **"\n"**);  
 **return** ticket;  
 }).forEach(listTickets::add);  
 } **protected void** changeQuestionsTicket(Ticket<Question2> ticket) {  
 List<Question2> questions = ticket.getQuestions();  
  
 **for** (**int** i = 0; i < questions.size(); i++) {  
 **int** level = **rangeQuest**.get(i);  
 WrapperList<Question2> wrapListQ = **mapWrapListQuestGroupByLevel**.get(level);  
 *// I must search question from child ticket, who level equals with question from list questions by level* **int** findIndex = (Methods.*findQuestByLevel*(questions, level));  
 **if** (findIndex < 0) {  
 **throw new** RuntimeException(**"find index < 0 probably into list child questions was reset level: "** + level  
 + **".\n No found level: "** + level);  
 }  
 Question2 questFromChildTick = questions.get(findIndex);  
 Question2 questFromList;  
 **if** (wrapListQ.hasNext()) { *// THis block code, designed in case free questions* questFromList = wrapListQ.current();  
 **if** (questFromList.getId().equals(questFromChildTick.getId())) { *// this related with the advent of logic tryReplaceQuest* questFromList = tryReplaceThisQuest(questFromList, wrapListQ); *// in case id-question from list == id question-replace* } **else** {  
 questFromList = wrapListQ.next();  
 }  
 } **else** {  
 questFromList = giveRepeatedQuest(level); *// I take only those questions that con be repeated* **if** (Objects.*isNull*(questFromList)) { *// in ase if questions with repeated is present* **if** (!**prop**.isUnique()) {  
 questFromList = tryReplaceThisQuest(questFromChildTick, wrapListQ);  
 } **else** {  
 **throw new** RuntimeException(**"Compose unique questions not possible"**); } } } Collections.*swap*(questions, findIndex, i); *// this invoked very very very IMPORTANT!* questions.set(i, questFromList); } }  
 **protected** Question2 tryReplaceThisQuest(Question2 rscReplQuest, WrapperList<Question2> wrapperList) {  
 **if** (wrapperList.isEmpty()) **throw new** RuntimeException(**"list is empty, by try replace question "**);  
 **if** (wrapperList.size() == 1) { **return** rscReplQuest; }  
 **if** (wrapperList.hasNext()) {  
 Collections.*rotate*(wrapperList.getList().subList(wrapperList.getCurIndex(), wrapperList.size()), -1);  
 **return** wrapperList.next(); }  
 **int** findIndex = Methods.*findQuest*(wrapperList, rscReplQuest, Comparator.*comparing*(Question2::getId));  
 **if** (findIndex < 0) { **throw new** NoSuchElementException(TicketsGeneratorWayImpl2.**class** + **" no find element by id: "** + rscReplQuest); }  
 Collections.*swap*(wrapperList.getList(), 0, findIndex);  
 *// if wrapperList.hasNext == false* Collections.*shuffle*(wrapperList.getList().subList(1, wrapperList.size()));  
 Collections.*rotate*(wrapperList.getList(), -1);  
 wrapperList.resetCurIndex();  
 **return** wrapperList.next(); }  
  
 **protected** Question2 giveRepeatedQuest(**int** level) {  
 WrapperList<Question2> wrapListRepeatQ = **mapWrapListQuestRepeatedGroupByLevel**.get(level);  
 **if** (!wrapListRepeatQ.isEmpty()) {  
 **if** (!wrapListRepeatQ.hasNext()) wrapListRepeatQ.resetCurIndex();  
  
 Question2 q = wrapListRepeatQ.current();  
 q.setRepeat(q.getRepeat() - 1);  
  
 **if** (q.getRepeat() == 0) {  
 wrapListRepeatQ.remove(); *// removed quest with prop:repeat == 0  
 // but curStatePosRepeatQuest left unchanged* } **else** {  
 wrapListRepeatQ.next();  
 }  
 **return** q;  
 }  
 **return null**;  
 }  
  
 **protected** List<TicketNode> initTicketChildrenFromParent(List<Ticket<Question2>> listTicketsParent) {  
 **int** quantityTickets = listTicketsParent.size();  
 **int** remainQuantityTickets = **prop**.getQuantityTickets() - quantityTickets;  
 **int** fullPass = remainQuantityTickets / quantityTickets;  
 **int** remainsChildrens = remainQuantityTickets - fullPass \* quantityTickets;  
 **int** size = Math.*min*(quantityTickets, remainQuantityTickets);  
 List<TicketNode> listTicketNode = **new** ArrayList<>(size);  
 **for** (**int** i = 0; i < size; i++) {  
 **int** nNodes = fullPass;  
 **if** (i < remainsChildrens) {  
 nNodes++;  
 } listTicketNode.add(TicketNode.*of*(listTicketsParent.get(i), nNodes));  
 } **return** listTicketNode;  
 }  
 **private static class** Methods {  
 **static** <T> **int** findQuest(List<T> list, T elem, Comparator<T> comparator) {  
 **for** (**int** j = 0; j < list.size(); j++) {  
 **if** (comparator.compare(list.get(j), elem) == 0) {  
 **return** j;  
 } }  
 **return** -1;  
 }  
 **static** <T **extends** Question2> **int** findQuestByLevel(List<T> list, **int** level) {  
 **for** (**int** j = 0; j < list.size(); j++) {  
 **if** (list.get(j).getLevel() == level) {  
 **return** j; } }  
 **return** -1; } }  
 **protected static class** TicketNode {  
 **private** Ticket<Question2> **ticket**;  
 **private** LinkedList<TicketNode> **childrenNodes**;  
 **private** TicketNode() { **childrenNodes** = **new** LinkedList<>(); }  
 **private** TicketNode(Ticket<Question2> ticket) { **this**(); **this**.**ticket** = ticket; }  
 **private** TicketNode(Ticket<Question2> ticket, LinkedList<TicketNode> childrenNodes) {  
 **this**.**ticket** = ticket; **this**.**childrenNodes** = childrenNodes; }  
 **public** Ticket<Question2> getTicket() { **return ticket**; }  
 **public** List<TicketNode> getChildrenNodes() { **return childrenNodes**; }  
 **public void** setTicket(Ticket<Question2> ticket) **this**.**ticket** = ticket; }  
 **public void** setChildrenNodes(LinkedList<TickeNode> childrenNodes) {  
 **this**.**childrenNodes** = childrenNodes; } **public int** getQuantityChildrenNodes() {  
 **return childrenNodes**.size(); }  
 **public boolean** addNode(TicketNode node) {  
 **return childrenNodes**.add(node); }  
 **public static** TicketNode of(Ticket<Question2> ticket) { **return new** TicketNode(ticket); }  
 **public static** TicketNode of(Ticket<Question2> ticket, **int** nChildNodeFromParent) {  
 TicketNode parNode = **new** TicketNode(ticket);  
 parNode.initChNodesFromParentClone(nChildNodeFromParent); **return** parNode; }  
 **public static** TicketNode of(Ticket<Question2> ticket, LinkedList<TicketNode> childrenNodes) { **return new** TicketNode(ticket, childrenNodes); }  
 **private void** initChNodesFromParentClone(**int** nChNoes) { **for** (**int** i = 0; i < nChNodes; i++) {  
 TicketNode chNode = TicketNode.*of*(**ticket**.clone()); **childrenNodes**.add(chNode); } } }}

# ПРИЛОЖЕНИЕВ*.* ТРЕБОВАНИЕ К СТРУКТУРНОМУ СОДЕРЖИМОМУ ФАЙЛА ВОПРОСОВ

Для того чтобы из файла считать список вопросов, в файле требуется определить элемент: <S> – определяет список вопросов, который оформляется по определенным требованиям (смотрите рисунок 5.6).

Он имеет начальный тег: <S> и конечный тег: <S/>.

**Так же к начальному тегу можно применить следующую атрибуты:**

* “l” – атрибут указывает сложность. Тип данных – целочисленное число;
* “n” – атрибут указывает название темы. Тип данных – строка;
* “r” – атрибут указывает число повторений. Тип данных – целочисленное число;

**Конструкция по размещению атрибутов внутри начального тега** выглядит следующим образом:

<S> n=Программирование систем; l=23; r = 3;>

Указание этих атрибутов не является обязательной, в случаи если используется способ формирования билетов «Сплошным списком», поэтому можно указать просто начальный тег: <S>. Однако если способ формирования билетов является «Группировка по сложности» или «Группировка по темам», то указание этих атрибутов необходимо.

**Требования к оформлению атрибутов**

* Сначала указывается *название* атрибут;
* После *названия* атрибута ожидается символ “**=**”, Между *названием* атрибута и символом “=” могут идти пробелы;
* После символа “=” ожидается *значение* атрибута, тип которого определяется самим атрибутом. Между символом “=” и *значением* могут идти пробелы;
* После *значения* атрибутаожидается символ “;”. Между *значением* и символом “;” могут идти пробелы;
* Внутри начального тега могут содержаться *все* или *некоторые* атрибуты, в *любом порядке* их определения. Атрибуты, указанные внутри начального тега <S>…> будут распространяться на атрибуты всех вопросов внутри этого списка.

**Требования к оформлению абзаца на наличие тега**

* Теги пишется заглавными буквами;
* Абзац должен быть выравнен по центру;
* Абзац не должен содержать ничего другого, кроме тега <S> или <S/>

**Требования к оформлению списка вопросов**

* Список вопросов должен находится между начальным тегам <S> и конечным тегом <S/>;
* Каждый вопрос должен быть нумерован;
* После объявления открывающего тега <S> с нового абзаца должен идти нумерованный список вопросов, после которого с нового абзаца должен идти закрывающий тег <S/>;

**Есть возможность указать эти же атрибуты для конкретного вопроса.** Для этого в *начале текстового содержимого* нумерованного вопроса нужно указать элемент: **{ }**, внутри которого можно указать все описанные ранее атрибуты.

В таком случаи значения атрибутов вопроса будут перезаписаны на значения указанных атрибутов, внутри этой конструкции.

**Конструкция по размещению атрибутов внутри вопроса** выглядит следующим образом (полужирным выделено):

1. **{ l=23; r = 3; }**Какое-то содержимо вопроса….

**Содержимое вопроса.**

Как было сказано, вопрос может содержать как текстовую, так и графическую информацию: рисунки, уравнения, формулы.

К текстовому содержимому вопроса, могут быть применимы все виды форматирования: цвета, полужирный, курсив, зачеркивание, подчеркивание, надстрочный подстрочный текст, а также текстовые эффекты, такие как контур, тень, отражения и другие.

К рисунку могут быть применимы все режимы параметры разметки: в тексте, обтеканием текстом. В последнем случаи, рисунок будет относится к тому вопросу, к какому привязан он абзацу, а также иметь такое же расположение, какое было указана, при оформлении его в вопросе.

К вопросу будет относиться содержимое от начала нумерации до следующего вопроса либо до конечного тега.

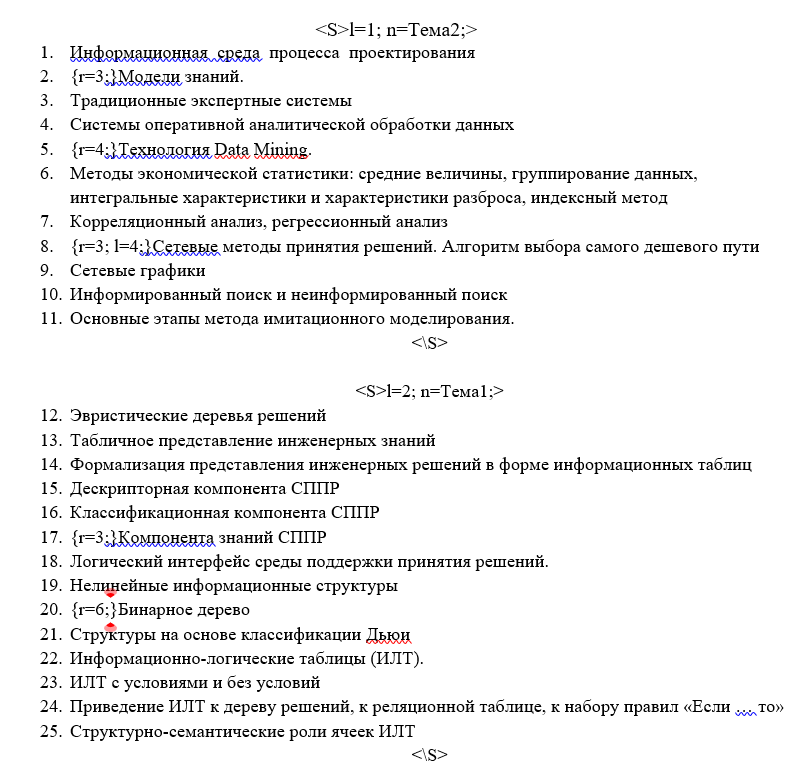


Рисунок B.1 – Пример оформленного файла