# **Федеральное государственное бюджетное профессиональное образовательное учреждение**

**Торжокский политехнический колледж**

# **Федерального агентства по государственным резервам**

**ОТЧЁТ**

# о зимней работе

**по профессиональному модулю МДК.01.01**

«**РАЗРАБОТКА ПРОГРАМНЫХ МОДУЛЕЙ**»

наименование профессионального модуля

## Специальность ***09.02.07 «Информационные системы и программирование»***

Период с «28» декабря 2024г. по «08» февраля 2025г.

## Выполнили студенты

Гаврильченко С.А., Баранов Д.В.

(ФИО, подпись)

|  |  |
| --- | --- |
| Преподаватель  Бойков А.М.  (ФИО, подпись) | оценка |

## 

2025

**Введение**

Современная образовательная система сталкивается с необходимостью оптимизации процессов, связанных с подготовкой и проведением аттестационных мероприятий. Одной из ключевых задач в этом контексте является разработка эффективных инструментов для создания экзаменационных материалов. Особую сложность представляет процесс формирования вариативных билетов, требующий соблюдения нескольких важных критериев: равномерного распределения вопросов по тематическим разделам, обеспечения уникальности содержания каждого билета и контроля достаточности вопросов для формирования требуемого количества вариантов.

Целью данной работы является разработка программного модуля для автоматизации процесса создания экзаменационных билетов.

Особенностью проекта является его ориентация на практическое применение в условиях реального учебного процесса. Программа разрабатывается с учетом специфики работы преподавателей, не обладающих специальной технической подготовкой, что делает её доступной для широкого внедрения в образовательных учреждениях различного уровня.

Методологической основой разработки послужили принципы объектно-ориентированного программирования, что позволило создать модульную систему с четким разделением функциональных обязанностей между компонентами. Реализация выполнена на платформе .NET с использованием языка C#, что обеспечивает надежность работы и возможность дальнейшего расширения функциональности.

В рамках работы были выполнены следующие этапы:

1. Анализ существующих проблем и формулировка требований к программе.
2. Проектирование модульной структуры приложения.
3. Реализация алгоритмов загрузки вопросов, генерации билетов и их сохранения.
4. Тестирование функциональности программы.

**Раздел 1 – Практическая значимость разработки** **ПО**

Проблема заключатся в том, что в настоящее время процесс составления экзаменационных билетов вручную является трудоемким, требует значительных временных затрат и подвержен ошибкам. Преподавателю приходится вручную выбирать вопросы из разных разделов, следить за тем, чтобы вопросы не повторялись, и проверять, хватает ли вопросов для формирования нужного количества билетов. Это особенно сложно при большом количестве билетов или при необходимости частого обновления вопросов.

Задачи:

1. Создание эффективной структуры для хранения вопросов по трем разделам: «знать», «уметь», «владеть».

2. Разработка логики для случайного выбора вопросов из каждого раздела с учетом уже выбранных.

3. Создание удобного интерфейса для ввода количества билетов и выбора файла для сохранения.

4. Реализация механизма проверки правильности составления вопросов и возможность добавления новых.

Сложности:

* Нехватка вопросов – при попытке создать много билетов система предупредит, если вопросов недостаточно, и предложит добавить новые.
* Уникальность вопросов в билетах – программа сама исключает повторы, но требует достаточного количества вариантов.
* Работа с файлами – нужно корректно загружать вопросы и сохранять билеты, чтобы не потерять данные.
* Простота использования – интерфейс должен быть интуитивным, даже для преподавателей без технических навыков.

Сейчас процесс формирования билетов происходит по опыту прошлых лет.

1. Составление вопросов:

- Преподаватель вручную создает список вопросов по каждому разделу («знать», «уметь», «владеть») и сохраняет их в текстовом файле или таблице.

2. Формирование билетов:

- Преподаватель вручную выбирает вопросы из каждого раздела и формирует билеты, следя за тем, чтобы вопросы не повторялись.

3. Проверка достаточности вопросов:

- Преподаватель вручную подсчитывает количество вопросов в каждом разделе и проверяет, хватает ли их для формирования нужного количества билетов.

4. Сохранение билетов:

- Преподаватель вручную сохраняет билеты в текстовый файл или распечатывает их.

5. Обновление вопросов:

- Если вопросов недостаточно, преподаватель вручную добавляет новые вопросы в файл и повторяет процесс формирования билетов.

**Раздел 2 – Пользовательские требования**

1. Пользователь должен иметь возможность ввести количество необходимых билетов.
2. Программа должна проверять, что введенное значение является целым числом (тип int).
3. Пользователь должен иметь возможность указать файл с вопросами, который будет использоваться для генерации билетов. Файл должен содержать вопросы в определенном формате. (Описано в 4 пункте.)

В программе предусмотрен сценарий указания файла с вопросами:

- При запуске программа запрашивает у пользователя путь к файлу с вопросами.

- Пользователь вводит путь к файлу с вопросами в консоли при запуске программы или через меню добавления вопросов.

- Программа проверяет существование файла и его соответствие требуемому формату:

1. если файл существует и соответствует требуемому формату, то вопросы успешно загружаются, пользователь в сою очередь получает сообщение в консоли об успешной загрузке файла;
2. если файл не существует, программа выводит сообщение об ошибке и предлагает ввести путь снова;
3. если файл имеет неверный формат, то программа выводит на консоль список ошибок и предлагает исправить файл;
4. если программе не удаётся получить доступ к файлу, то программа сообщает об этом пользователю через консоль о невозможности чтения файла (например, отсутствие прав).
5. Пользователь должен иметь возможность указать путь для сохранения сгенерированных билетов в формате .docx.

В программе предусмотрен сценарий указания файла с вопросами:

- После генерации билетов пользователю предлагается ввести путь для сохранения файла.

- Если расширение не указано, автоматически добавляется .docx.

Исключения:

1. если пользователь оставил путь к файлу пустым, то программа выведет ошибку на консоль “ Путь не может быть пустым!”;
2. если файл успешно сохранён, но на устройстве, например по умолчанию не назначено приложение для открытия .docx или этот файл уже открыт, пользователю высветится в консоли сообщение об ошибке “ Документ сохранён, но не удалось открыть его”;
3. если указан неверный путь к шаблону, шаблон повреждён или заблокирован другим процессом (все проблемы которые связаны с самим шаблоном), то пользователь получит сообщение в консоль об ошибке “ Ошибка при создании документа”.
4. Формат файлов:

- Формат файла с вопросами должен быть строго «.txt». Программа должна предоставить пример имени и расширения файла перед тем, как пользователь начнёт вводить путь к файлу (Например questions.txt).

- Формат строк: Каждая строка файла с вопросами представляет один вопрос и должна иметь следующий формат: <раздел> | <текст вопроса>

* <раздел>: Строка, указывающая раздел вопроса. Допустимые значения: “знать”, “уметь”, “владеть”. Регистр имеет значение.
* | (вертикальная черта): Разделитель между разделом и текстом вопроса. Важно использовать именно этот символ. От текста до черты и от черты до текста должен стоять пробел.
* <текст вопроса>: Строка, содержащая текст вопроса. Может содержать любые символы, за исключением символа |. Если этот символ содержится в тексте вопроса, такие вопросы следует заключать в двойные кавычки.

Пример такой строки:



1. Случайный отбор вопросов:

- Функция должна реализовать алгоритм для случайного выбора вопросов из каждой категории, гарантируя, что выбранные вопросы не повторяются в пределах одного билета.

- Обеспечить, чтобы каждый билет содержал ровно по одному вопросу из каждого раздела.

- Обеспечить, чтобы выбранные для генерации билета вопросы не повторялись в пределах всех сгенерированных билетов.

1. Проверка наличия достаточного количества вопросов:

- Перед созданием билетов необходимо проверить, достаточно ли вопросов в каждой категории для выполнения задания. (Достаточно - если введённое пользователем число билетов равно или меньше количества вопросов в каждом разделе, то программа выводи на консоль список готовых билетов. Недостаточно - если введённое пользователем число билетов больше количества вопросов в хотя бы в одном разделе, то программа просит пользователя ввести новый(-ые) вопрос(-ы) в раздел(-ы), где их недостаточно.) - Если вопросов оказывается недостаточно, система должна информировать пользователя об этом сообщением в консоли “Недостаточно вопросов!” .

- Если вопросов оказывается достаточно, система должна информировать пользователя об этом показав в консоли сгенерированные билеты.

1. Добавление новых вопросов:

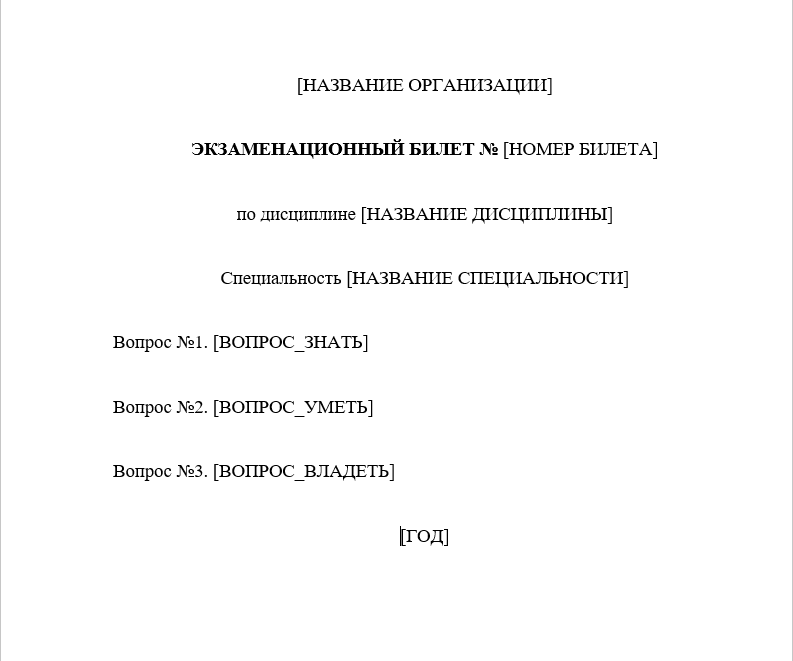
- В случае недостаточного количества вопросов для формирования билетов должна быть предусмотрена возможность ввода дополнительных вопросов вручную. Программа предлагает пользователю ввести столько вопросов в каждый раздел, чтобы их хватило для генерации хотя бы одного билета.

- Обновление исходного текстового файла с вопросами должно происходить автоматически по принципу нажатия клавиши Enter после ввода новых данных. Программа считывает вопросы, добавляя их в список файла, из которого ранее считывала вопросы для последующего повторного прочтения.

1. Сохранение и форматирование билетов:

- Сохранение сгенерированных экзаменационных билетов в документ Word по шаблону.

Пример:

****

**Раздел 3 – Проектирование модульной структуры программы**

В данной программе представлены 5 классов.

На рисунке 1 представлена диаграмма классов.

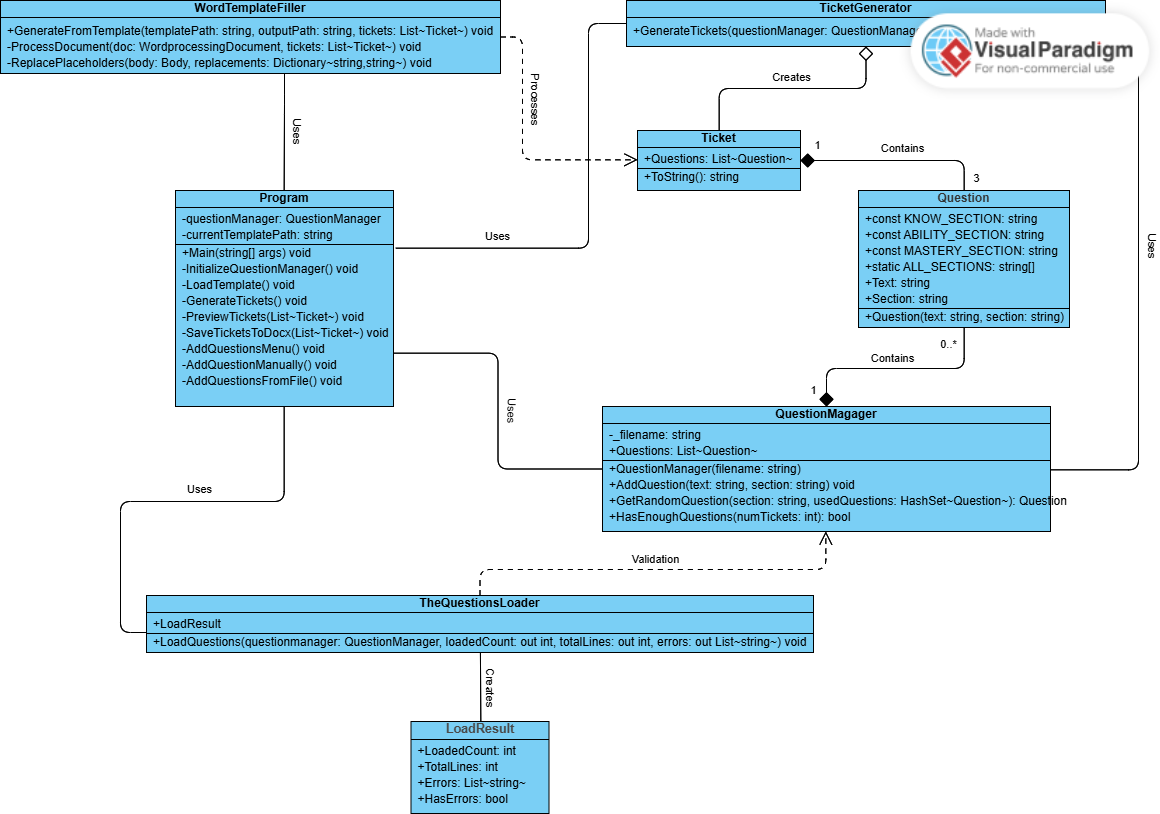


Рисунок 1 – Диаграмма классов

Класс Question отвечает за представление вопроса, т.е. представляет единичный вопрос для экзаменационного билета.

|  |
| --- |
| Question |
| public const string KNOW\_SECTION = "знать";  public const string ABILITY\_SECTION = "уметь";  public const string MASTERY\_SECTION = "владеть";  public static readonly string[] ALL\_SECTIONS = { KNOW\_SECTION, ABILITY\_SECTION, MASTERY\_SECTION };  public string Text { get; set; }  public string Section { get; set; } |
| public Question(string text, string section) |

Класс Ticket отвечает за представление билета со списком вопросов, т.е. представляет экзаменационный билет, который состоит из набора вопросов.

|  |
| --- |
| Ticket |
| private List<Question> Questions { get; set; } = new List<Question>(); |
| public override string ToString() |

Метод ToString() отвечает за представление объекта в виде строки. Метод можно переопределить в пользовательских классах, чтобы предоставить более информативное строковое представление объекта.

Класс QuestionManager управляет списком всех вопросов, загружает их из файла, добавляет новые вопросы и предоставляет возможность выбрать случайный вопрос из определенного раздела.

|  |
| --- |
| QuestionManager |
| public List<Question> Questions { get; set; } = new List<Question>();  private string \_filename; |
| public QuestionManager(string filename)  public void AddQuestion(string text, string section)  public Question GetRandomQuestion(string section, HashSet<Question> usedQuestions)  public bool HasEnoughQuestions(int numTickets) |

Метод AddQuestion() добавляет в файл с вопросами вопросы, которые ввёл пользователь при недостаточном количестве вопросов для формирования билета.

Метода HasEnoughQuestions() проверяет достаточно ли вопросов для генерации билетов, т.е. он сравнивает количество вопросов в каждом разделе с заданным пользователем необходимым количеством билетов.

Класс TicketGenerator отвечает за создание экзаменационных билетов на основе имеющихся вопросов.

|  |
| --- |
| TicketGenerator |
| - |
| public List<Ticket> GenerateTickets(QuestionManager questionManager, int numTickets) |

Метод GenerateTickets()

Входные данные:

1. Ссылка на объект класса QuestionManager. Этот объект содержит список всех доступных вопросов, загруженных из файла, а также предоставляет методы для работы с этими вопросами (выбор случайного, проверка наличия достаточного количества и т.д.).

Метод GenerateTickets() использует этот объект для:

Проверки, достаточно ли вопросов для создания заданного количества билетов (через вызов questionManager.HasEnoughQuestions(numTickets)).

Получения случайных вопросов из разных разделов для каждого билета (через вызов questionManager.GetRandomQuestion(section, usedQuestions)).

1. numTickets (тип: int): Целое число, указывающее, сколько экзаменационных билетов необходимо сгенерировать.

Метод GenerateTickets() использует это число для определения количества итераций цикла, в котором создаются билеты.

Выходные данные:

List<Ticket> (Список объектов типа Ticket) или null

Если метод успешно сгенерировал все билеты, он возвращает список, содержащий numTickets объектов типа Ticket. Каждый объект Ticket представляет собой экзаменационный билет, содержащий по одному вопросу из каждого раздела (“знать”, “уметь”, “владеть”).

Если метод не смог сгенерировать все билеты (например, из-за недостатка вопросов в каком-либо разделе), он возвращает null.

Класс TheQuestionsLoader отвечает за загрузку вопросов из файла.

|  |
| --- |
| TheQuestionsLoader |
| public struct LoadResult |
| static public LoadResult LoadQuestions(QuestionManager questionmanager) |

Метод LoadQuestions() осуществляет загрузку вопросов из файла.

Осуществляет обработку ошибок:

* Если файл с вопросами не найден, пользователю на консоль выводится сообщение об ошибке “Ошибка: файл «имя\_файла» не найден!” и метод завершает работу.
* Обрабатываются исключения при чтении файла (например, проблемы с доступом). На консоль выводится сообщение об ошибке “Ошибка при чтении файла!” и метод завершает работу.
* Если при чтении файла обнаруживаются пустые строки, метод их пропекает и выводит сообщение об ошибке “Пустая строка”.
* Если количество частей строки не равно двум, выводится сообщение об ошибке формата строки в консоль.
* Если количество частей в строке равно двум и одна из частей пуская, выводится сообщение об ошибке в консоль “Пустое значение в строке”.

Итоговый отчет: В конце выполнения метода LoadQuestions() в консоли отображается информация о количестве успешно загруженных вопросов относительно общего числа строк в файле.

Класс WordTemplateFiller отвечает за генерацию экзаменационных билетов в формате Word (.docx) на основе шаблона.

|  |
| --- |
| WordTemplateFiller |
| - |
| public static void GenerateFromTemplate(string templatePath, string outputPath, List<Ticket> tickets)  private static void ProcessDocument(WordprocessingDocument doc, List<Ticket> tickets) private static void ReplacePlaceholders(Body body, Dictionary<string, string> replacements) |

Метод GenerateFromTemplate(), который загружает шаблон Word, заполняет его данными из списка билетов и сохраняет результат в новый файл.

Аргументы, которые принимает метод:

String templatePath - Путь к файлу шаблона (\*.docx).

String outputPath - Путь, куда сохранить заполненный документ.

List<Ticket>Tickets - Список билетов, каждый из которых содержит вопросы для подстановки.

Работает по следующему принципу:

1. Чтение шаблона → загружает файл в память (MemoryStream), чтобы избежать блокировки.
2. Обработка документа → вызывает ProcessDocument, который заменяет плейсхолдеры.
3. Сохранение результата → записывает измененный документ в outputPath.

Метод ProcessDocument(), который обрабатывает документ Word, заменяя плейсхолдеры на данные из билетов и добавляя разрывы страниц между ними.

Аргументы, которые принимает метод:

WordprocessingDocument doc - Открытый документ Word для редактирования.

List<Ticket>Tickets - Список билетов с вопросами.

Работает по следующему принципу:

1. Очистка документа → удаляет все элементы из тела документа.
2. Копирование шаблона → вставляет шаблон для каждого билета.
3. Замена плейсхолдеров → вызывает ReplacePlaceholders для подстановки данных.
4. Добавление разрывов страниц → вставляет разрыв после каждого билета (кроме последнего).

Метод ReplacePlaceholders(), который заменяет плейсхолдеры в документе на реальные значения из словаря.

Аргументы, которые принимает метод:

Body body - Тело документа Word (OpenXml).

Dictionary<string, string> replacements - Словарь вида { "[ПЛЕЙСХОЛДЕР]", "значение" }.

Работает по следующему принципу:

* Ищет все текстовые элементы (Text) в документе.
* Если находит плейсхолдер (например, [НОМЕР БИЛЕТА]), заменяет его на соответствующее значение.

**Раздел 4 – Описание алгоритмов программы.**

Метод LoadQuestions() отвечает за чтение вопросов из файла и создания объектов Question (вопрос), которые впоследствии добавляет в список Questions (вопросы).

Работает по следующему принципу:

* Читает вопросы из указанного файла.
* Разделяет каждую строку на раздел и текст вопроса, используя символ |.
* Создает объекты Question и добавляет их в список Questions.
* Очищает список Questions перед загрузкой новых вопросов

Метод AddQuestion(string text, string section) отвечает за добавление нового объекта Question (вопрос).

Работает по следующему принципу:

* Создает новый объект Question.
* Добавляет новый вопрос в список Questions.
* Сохраняет вопрос в файл

На рисунке 2 представлена блок-схема метода LoadQuestions().

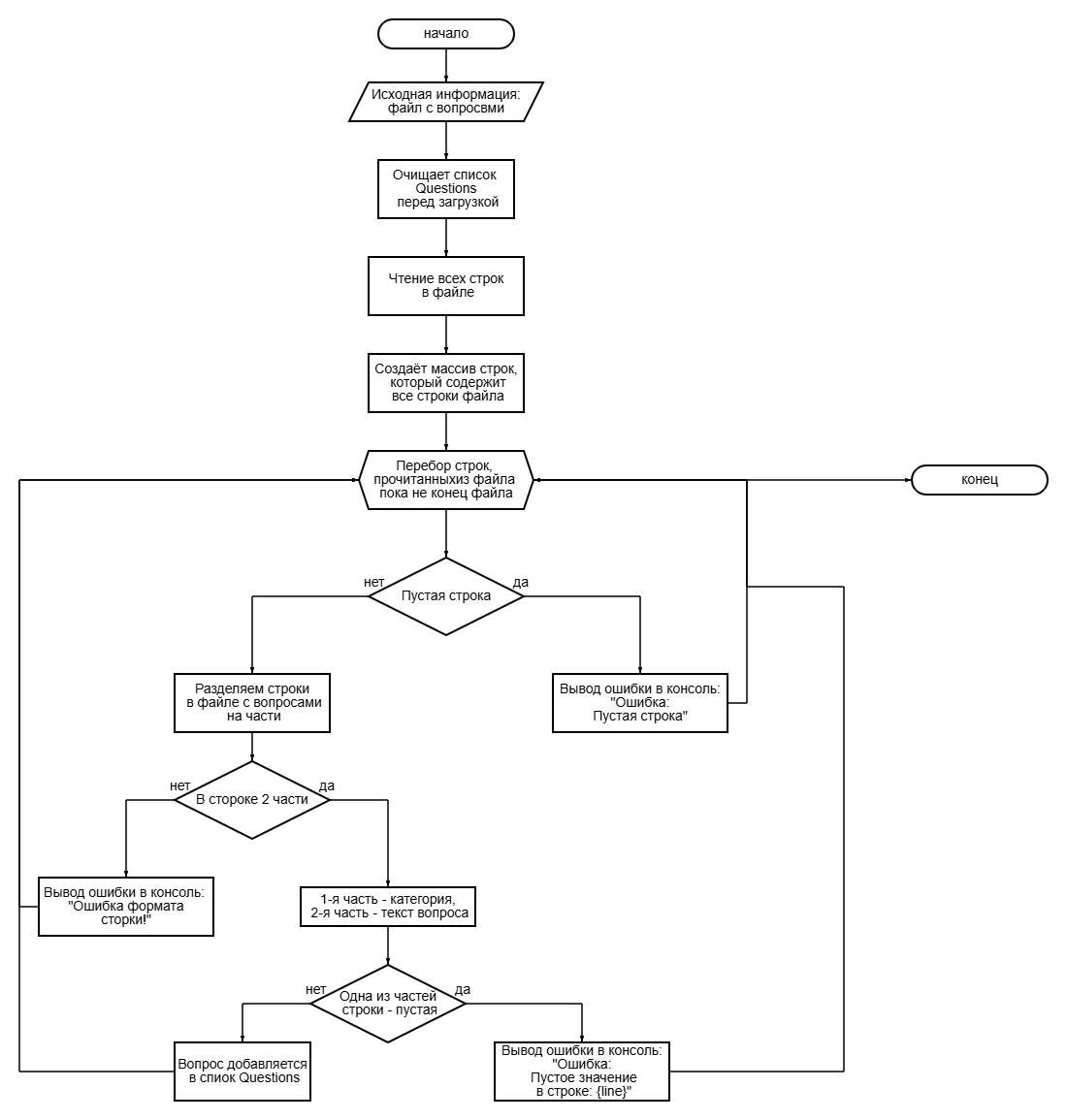


Рисунок 2 – Блок-схема метода LoadQuestions()

На рисунке 2 представлена блок-схема метода AddQuestion().

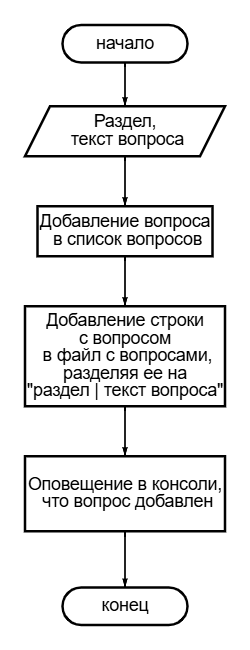
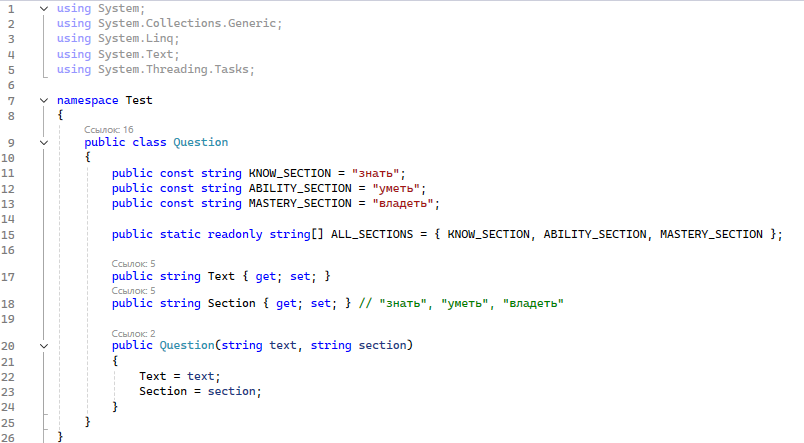


Рисунок 2 – Блок-схема метода AddQuestion().

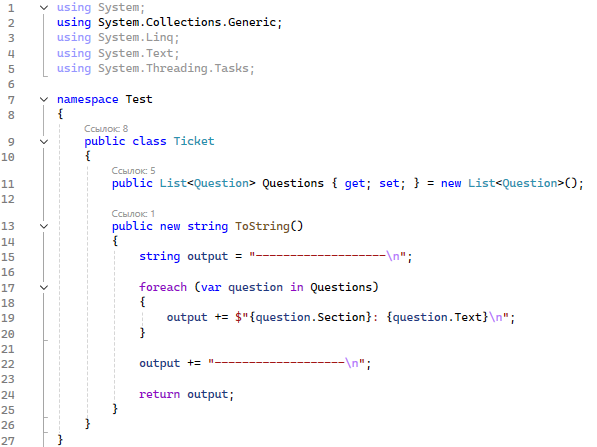
**Раздел 5 - Разработка проекта Visual Studio C#.**

На скриншоте 3 представлен код класса Question.

****

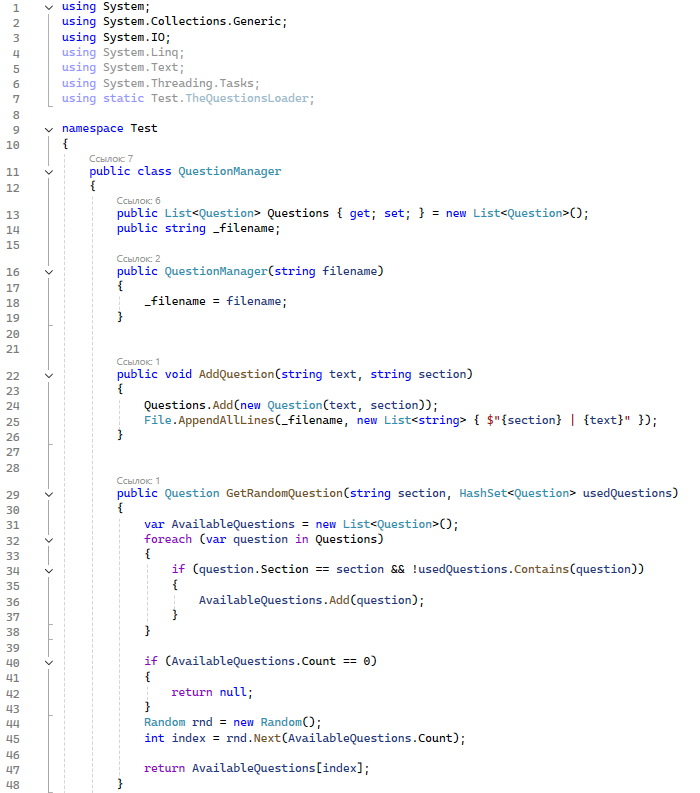
Скриншот 3 - Код класса Question

На скриншоте 4 представлен код класса Ticket.

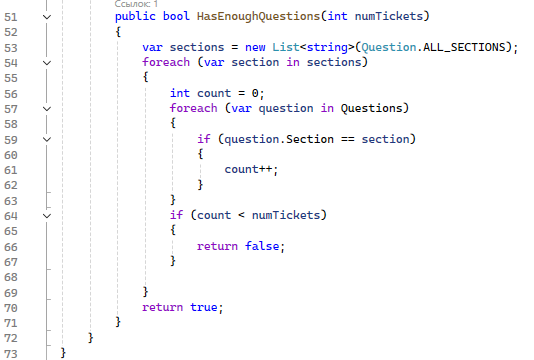
****

Скриншот 4 - Код класса Ticket

На скриншоте 5 и 6 представлен кода класса QuestionManager.

****

Скриншот 5 - Код класса QuestionManager



Скриншот 6 - Код класса QuestionManager

На скриншоте 7 представлен кода класса TicketGenerator.

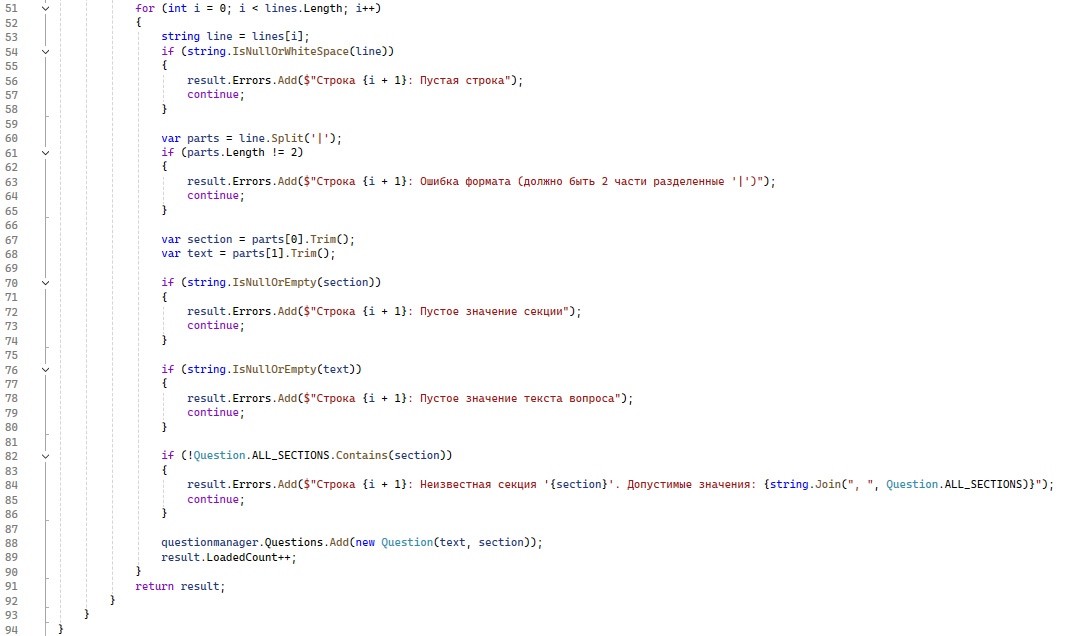
****

Скриншот 7 - Код класса TicketGenerator

На скриншоте 8 и 9 представлен кода класса TheQuestionsLoader.

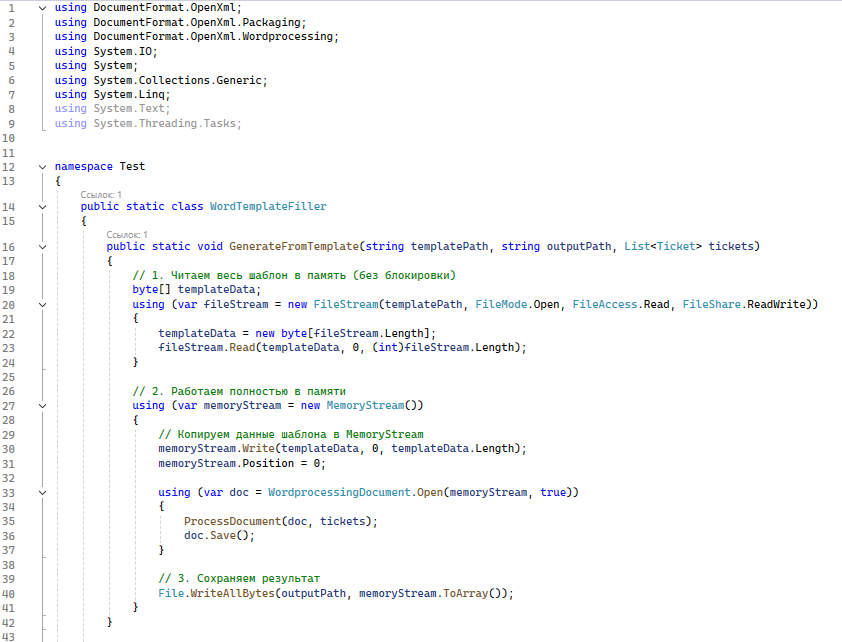
****

Скриншот 8 - Код класса TheQuestionsLoader

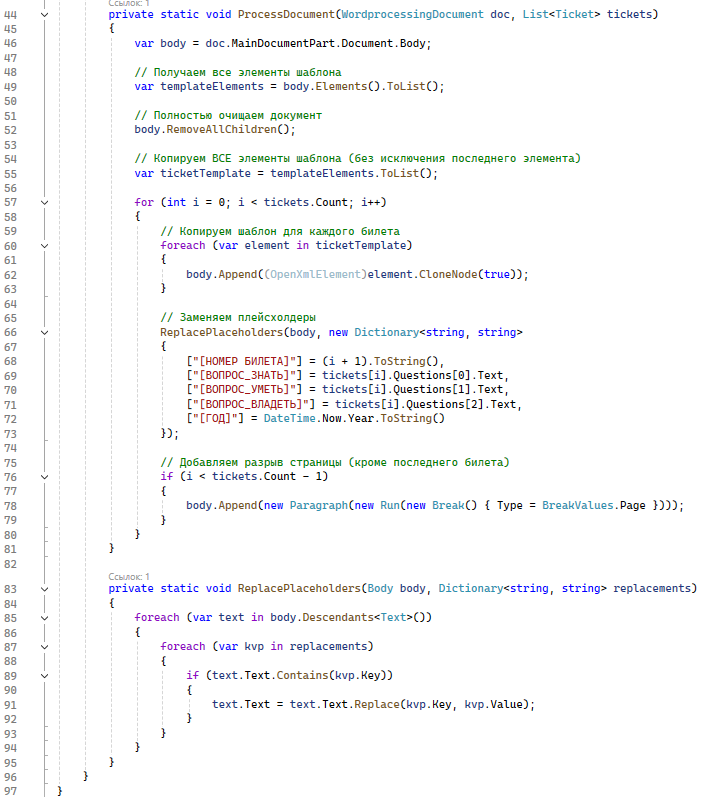


Скриншот 9 - Код класса TheQuestionsLoader

На скриншоте 10 и 11 представлен кода класса WordTemplateFiller.



Скриншот 10 - Код класса WordTemplateFiller



Скриншот 11 - Код класса WordTemplateFiller

На скриншоте 12, 13, 14, 15, 16, 17 представлен кода класса Program.



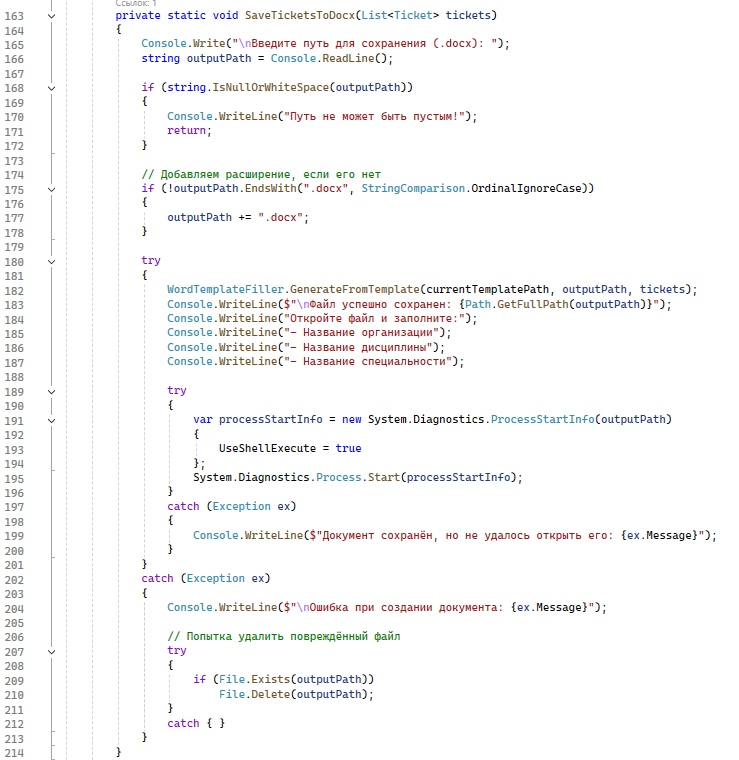
Скриншот 12 - Код класса Program



Скриншот 13 - Код класса Program



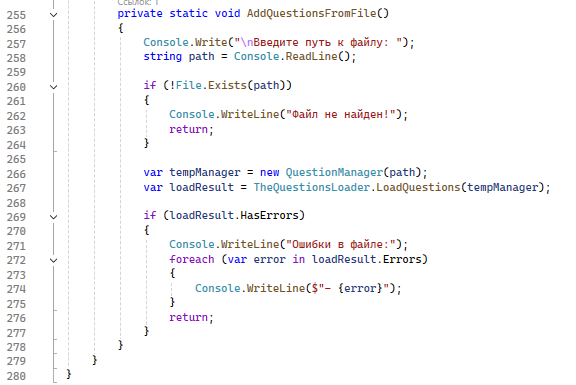
Скриншот 14 - Код класса Program



Скриншот 15 - Код класса Program



Скриншот 16 - Код класса Program



Скриншот 17 - Код класса Program