Федеральное государственное бюджетное профессиональное образовательное учреждение

Торжокский политехнический колледж

Федерального агентства по государственным резервам

Индивидуальное задание по МДК.01.01

«Разработка программных модулей»

Выполнил студент группы П-30

Гаврильченко С.А.

Баранов Д.М.

2025

**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. Интерфейс пользователя:

- Пользователь должен иметь возможность ввести количество необходимых билетов.

- Программа должна проверять, что введенное значение является целым числом (тип int).

- Предусмотреть возможность выбора файла для сохранения билетов и файла с вопросами, с примером как должен выглядеть файл с необходимым для работы программы расширением (например, questions.txt).

2. 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, кроме символа |.

Пример такой строки:



3. Случайный отбор вопросов:

- Модуль должен 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.

4. 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 совпадает с количеством вопросов в каждом разделе, то программа выводи на консоль список готовых билетов. Недостаточно - если введённое пользователем число билетов не совпадает с количеством 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.

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

5. Добавление новых вопросов:

- В случае нехватки вопросов должна быть предусмотрена возможность ввода дополнительных вопросов вручную. Если вопросов не хватает, программа предлагает пользователю ввести столько вопросов в каждый раздел, чтобы их хватало для генерации хотя бы одного билета.

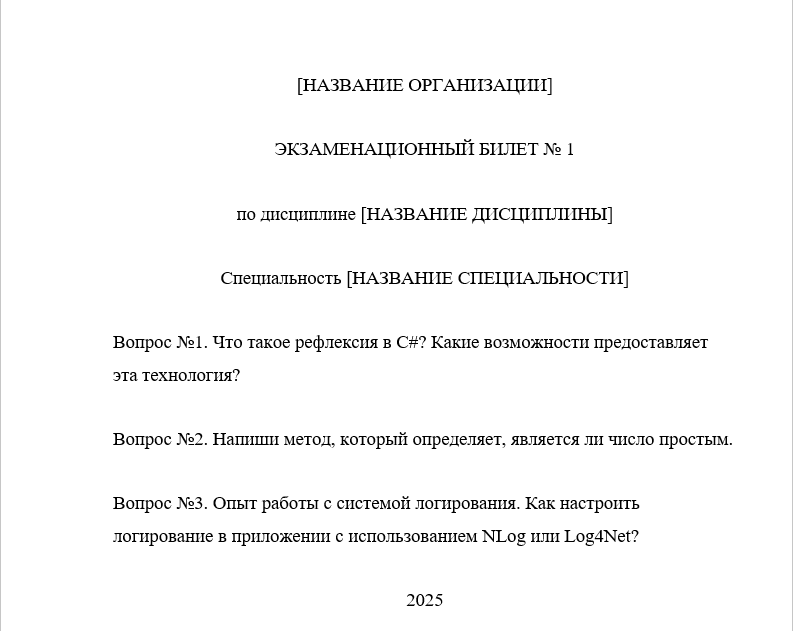
- Обновление исходного текстового файла с вопросами должно происходить автоматически по принципу нажатия клавиши Enter после ввода новых данных. Программа считывает вопросы, добавляя их в список файла, из которого ранее считывала вопросы для последующего повторного прочтения.

6. Сохранение и форматирование билетов:

- Сохранение сгенерированных экзаменационных билетов в документ Word должно быть реализовано с учётом формата, понятного пользователю. («Понятный» формат для сохранения экзаменационных билетов в документ Word подразумевает структуру, которая будет легко восприниматься пользователем без необходимости использования специальных программ для просмотра содержимого. Например: .docx)

- Каждый билет должен быть четко структурирован, чтобы пользователь мог легко его читать. (Четкая структура билета означает, что информация представлена таким образом, чтобы её было удобно воспринимать и быстро находить нужные сведения.)

Пример:



7. Кросс-платформенность:

- Модуль должен быть разработан с учётом возможности использования на различных операционных системах.

**Раздел 3 – Проектирование модульной структуры программы**

В данной программе представлены 5 классов.

На рисунке 1 представлена диаграмма классов.

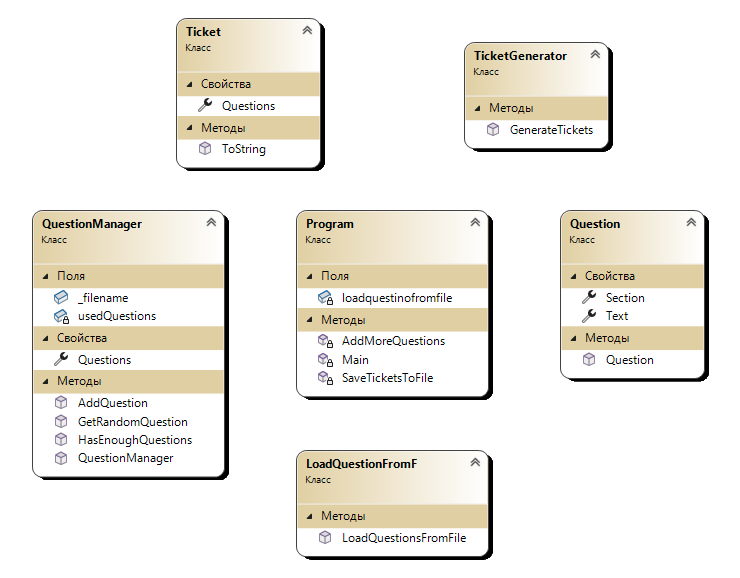


Рисунок 1 – Диаграмма классов

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

|  |
| --- |
| Question |
| public 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() |

Метода 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 |
| private List<Question> usedQuestions = new List<Question>();  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, List<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 |
| - |
| static public void LoadQuestions(QuestionManager questionmanager) |

Метод LoadQuestions() выполняет следующие действия:

1. Очистка списка: Сначала очищается существующий список вопросов вызовом questionmanager.Questions.Clear(), чтобы избежать дублирования данных при повторной загрузке.
2. Чтение строк из файла: Метод считывает все строки из файла, указанного в переменной \_filename объекта questionmanager, используя функцию File.ReadAllLines.
3. Цикл по строкам:
   * Для каждой строки происходит разделение её на части по символу | методом Split.
   * Если количество частей равно двум (if (parts.Length == 2)), то предполагается, что первая часть — это раздел (section), а вторая — текст вопроса.
   * Эти данные используются для создания нового объекта Question и добавления его в список вопросов.
4. Обработка ошибок:
   * Если количество частей не равно двум, выводится сообщение об ошибке формата строки в консоль.
5. Итоговый отчет: В конце выполнения метода в консоли отображается информация о количестве успешно загруженных вопросов относительно общего числа строк в файле.

**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().

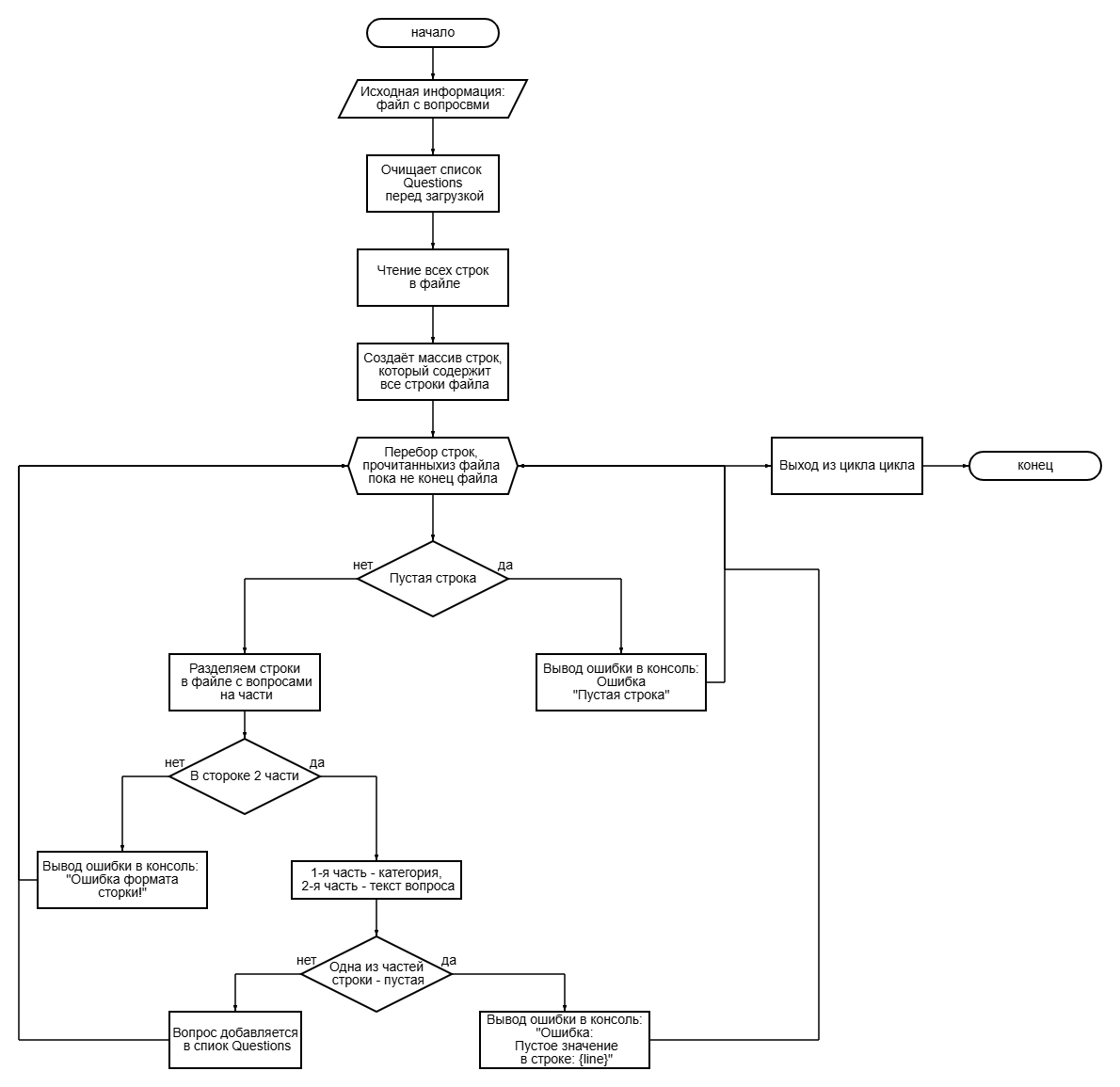


Рисунок 2 – Блок-схема метода LoadQuestions()

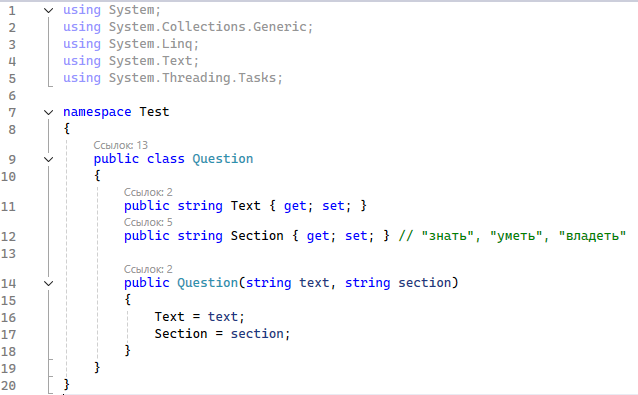
На рисунке 2 представлена блок-схема метода AddQuestion().



Рисунок 2 – Блок-схема метода AddQuestion().

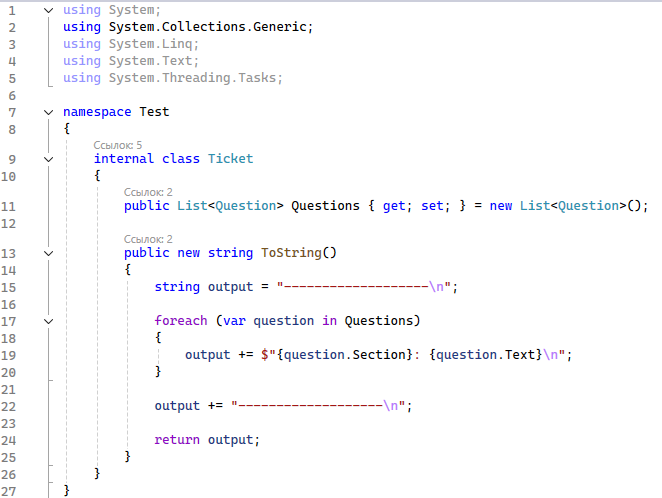
**Раздел 5 - Разработка проекта Visual Studio C#.**

На скриншоте 3 представлен код класса Question.

****

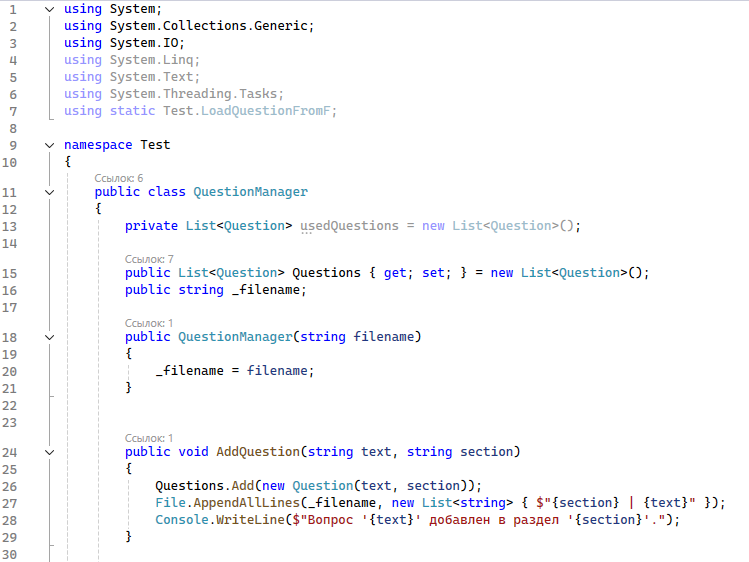
Скриншот 3 - Код класса Question

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 представлен кода класса LoadQuestionFromF.

****

Скриншот 8 - Код класса LoadQuestionFromF

На скриншоте 9 и 10 представлен кода класса Program.

****

Скриншот 9 - Код класса Program



Скриншот 10 - Код класса Program