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 ~~программы~~** **разработки** **ПО**

**То, что здесь описано, должно относиться к разделу №2 – требования к ПО. В разделе №1 описывается общая проблема, которая рассматривается при разработке ПО: какие задачи, сложности есть сейчас и почему данное ПО может быть востребовано, как осуществляются процессы сейчас, когда данного ПО нет.**

Требования к программному модулю:

1. Структура хранения вопросов:

- Необходимо создать структуру данных для хранения вопросов в трех разделах: «знать», «уметь», «владеть».

- Каждый вопрос должен быть уникальным и иметь идентификатор для удобного доступа.

**Требования такого уровня необходимо описывать в разделе №3 с привязкой к конкретным классам: их полям и методам.**

**Раздел №2 должен включать описание требований к тем задачам, которое должно позволить решить ПО пользователям, должно быть понятно как пользователю, так и разработчикам, отвечать на вопросы: какую задачу реальную позволит решить функция X и как этой функцией можно воспользоваться в самой программе.**

**Ключевое свойство требования – по описанию можно однозначно проверить, что программа соответствует данному требованию**

2. Интерфейс пользователя:

- Пользователь должен иметь возможность ввести количество необходимых билетов. (необходимо расширить требование проверкой вводимой пользователем информации, а также указать требование к тому, каким образом пользователь должен осуществить ввод)

- Предусмотреть возможность выбора файла для сохранения билетов и файла с вопросами. (дополнить описание требования: какой файл, как указывать)

3. Описать отдельное требование к формату файлов с исходной информацией о вопросах и разделах в нем и привести примеры содержимого такого файла.

3.1 Как подпункт описать требование к проверке формата этого файла и реакции программы на указанный пользователем файл несоответствующей структуры.

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. (требуется конкретно расшифровать – что значит «достаточно» и реакцию программы на «достаточно» и «недостаточно»).

- Если вопросов оказывается недостаточно, система должна информировать пользователя об этом. (а если достаточно? Каким образом должно осуществляться информирование?)

5. Добавление новых вопросов:

- В случае нехватки??? вопросов должна быть предусмотрена возможность ввода дополнительных вопросов вручную. (описать, как этот процесс должен происходить при использовании программой пользователем)

- Обновление исходного текстового файла с вопросами должно происходить автоматически после ввода новых данных. (в какой момент времени? По какому принципу?)

6. Сохранение и форматирование билетов:

- Сохранение сгенерированных экзаменационных билетов в текстовом файле должно быть реализовано с учётом формата, понятного пользователю.

(требуется описать этот «понятный» формат и привести примеры выходного файла)

- Каждый билет должен быть четко структурирован, чтобы пользователь мог легко его читать. (описать, что значит четко структурирован, привести пример, как должен выглядеть билет)

~~7. Обработка ошибок:~~

~~- Модуль должен обрабатывать возможные ошибки, такие как неверный ввод пользователя, отсутствие доступа к файлам и т.д.~~

~~- Предусмотреть возможность повторного запроса ввода в случае ошибок.~~

Обработку ошибок описывать индивидуально в рамках каждого требования

8. Кросс-платформенность:

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

Проблемы, которые может решить программа:

1. Автоматизация процесса создания экзаменационных билетов:

- Программа значительно упрощает подготовку билетов, избавляя преподавателя от необходимости вручную отбирать вопросы.

2. Сокращение времени на составление билетов:

- Быстрое и эффективное генерирование экзаменационных билетов позволяет преподавателю сэкономить время и сосредоточиться на других аспектах подготовки к экзамену.

3. Предотвращение повторения вопросов:

- Алгоритм обеспечивает отсутствие дублирующихся вопросов в билетах.

4. Разбиение по разделам:

- Возможность выбора вопросов по различным разделам помогает равномерно оценить знания студентов и разнообразит экзаменационный процесс.

5. Генерация случайных вопросов:

- Студенты не могут предугадать, какие вопросы будут в экзаменационных билетах, что увеличивает объективность оценки.

6. Сохранение и управление вопросами:

- Программа дает возможность дополнительно вводить и обновлять вопросы, что позволяет поддерживать актуальность содержания экзамена.

7. Уведомление об недостатке вопросов:

- Программа информирует преподавателя о необходимости добавления новых вопросов.

8. Легкость в экспорте данных:

- Возможность сохранения сгенерированных билетов в текстовый файл позволяет преподавателю легко распечатать.

9. Гибкость в настройках:

- Пользователь может самостоятельно определить количество билетов, что позволяет адаптировать программу под различные требования и условия.

Эту часть можно оставить в разделе №1, но ее нужно привязать к конкретным проблемам, которые возникают сейчас у преподавателя без данного ПО.

**См. комментарии по разделу выше**

**~~Раздел 2 – Функциональные требования~~**

1. ~~Пользователь определяет и вводит желаемое количество экзаменационных билетов.~~

~~2. Программа автоматически выбирает случайный вопрос из трех разделов: «знать», «уметь», «владеть» для создания билета.~~

~~3. Перед созданием билетов программа проверяет, достаточно ли вопросов в каждом разделе для запрашиваемого количества билетов.~~

~~4. Если вопросов недостаточно, программа уведомляет пользователя об этом и указывает какие разделы требуется дополнить.~~

~~5. Программа предлагает пользователю ввести дополнительные вопросы для каждого из разделов, где не хватает материала, с последующим обновлением исходного текстового файла.~~

~~6. После генерации экзаменационных билетов программа сохраняет их в текстовый файл, имя и местоположение которого указывает пользователь.~~

~~7. Программа гарантирует, что в каждом из сгенерированных билетов вопросы из разных разделов не повторяются.~~

~~8. Программа отображает сгенерированные билеты непосредственно в консоли для предварительного просмотра пользователем перед сохранением.~~

**Раздел 3 – Проектирование модульной структуры программы**

**В разделе должна быть представлена сначала диаграмма классов**

В данной программе представлены 4 класса.

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; } |
| - |

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

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

Небезопасно иметь прямой доступ к полю вопросов через свойство. Необходимо сделать приватное поле с методами чтения и записи информации в это поле.

Не описана функциональная значимость метода ToString()

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

Класс перегружен. Загрузку информации из файла необходимо отнести к отдельному классу, который в том числе займется проверкой формата файла. (см. выше комментарий синего требования 3 и 3.1)

|  |
| --- |
| QuestionManager |
| \_filename = filename;  LoadQuestionsFromFile(); |
| public void LoadQuestionsFromFile()  public void AddQuestion(string text, string section)  public Question GetRandomQuestion(string section, List<Question> usedQuestions)  public bool HasEnoughQuestions(int numTickets) |

Метод LoadQuestionsFromFile не принимает никакой информации и никакой информации не возвращает. Непонятно, из какого файла будет производиться загрузка информации. Вижу поле \_fileName, но оно не указано как поле: у него нет типа, а также метода, который задаст значение этому полю.

Метод AddQuestion позволяет добавить вопрос, но в классе нет структуры данных, которая позволит сохранить эти данные.

GetRandomQuestion требует в качестве аргумента список вопросов. Очевидно, список вопросов должен быть полем класса QuestionManager, тогда необходимость в этом аргументе отпадет.

Не описан функционал метода HasEnoughQuestions

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

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

**Описать входные и выходные параметры данного метода**

**Раздел 4 – Описание алгоритмов программы.**

Метод LoadQuestionsFromFile() 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

Требуется также блок-схема