# Задание №6. Логика второстепенного окна

**Описание задания:**

В предыдущем задании была реализована логика главного окна, где функция добавления новых данных временно заменена на программную генерацию. Для добавления новых данных или редактирования ранее введенных согласно ТЗ нужно второе окно. В данном задании необходимо подготовить вторую форму для ввода и проверке введенных данных, и уже в следующем задании будет необходимо организовать передачу данных между окнами.

В ходе данного задания необходимо:

1. Реализовать инициализацию второго окна заранее подготовленными пользовательскими данными.
2. Реализовать валидацию пользовательского ввода.
3. Реализовать формирование объекта структуры данных на основе введенных пользователем данных.

В ходе работы не забывайте делать коммиты в репозиторий.

**Последовательность работы:**

1. Создайте новую ветку в локальном репозитории под названием «features/6_add_secondary_form_logic». Перейдите в новую ветку.
2. В классе Program проекта View замените в функции Main вызов конструктора главного окна MainForm на вызов конструктора второго окна ***Object***Form. Теперь при запуске приложения будет сразу открываться второе окно вместо главного, что ускорит отладку окна. (здесь и далее вместо ***Object*** необходимо подставить название структуры данных, соответствующей вашему варианту, например, Note, Contact, Movie и т.д.).
3. Во втором окне ***Object***Form добавьте закрытое поле ***_object*** типа ***Object***. При объявлении поля проинициализируйте его новым экземпляром класса.
4. В конструкторе ***Object***Form присвойте в экземпляр _object какие-либо данные.
5. Создайте закрытый метод UpdateForm(). Метод должен брать данные из поля _object и выводить их в соответствующие поля на форме.

6. Добавьте вызов метода UpdateForm в конце конструктора ***Object***Form() и запустите программу. Введенные программно данные должны отобразиться в элементах окна.

7. Добавьте обработчик события TextChanged для текстового поля TextBox, в котором должна выполняться валидация данных. Например, текстовое поле для фамилии контакта или текстовое поле для названия заметки.

8. В обработчике события добавьте код, который будет присваивать текст из текстового поля TextBox в соответствующее строковое свойство объекта _object. Поставьте точку останова в обработчике, запустите программу и убедитесь, что присвоение данных в свойство действительно происходит.

9. Как вы помните, реализация структур данных в одном из предыдущих заданий предполагала валидацию данных – при попытке присвоения некорректных данных в свойство, объект должен бросить исключение ArgumentException. Теперь здесь в форме необходимо организовать обработку исключений при присвоении данных.

10. Поместите код с присвоением текста из текстового поля в свойство _object в блок try, после блока try напишите блок catch для ArgumentException. В блоке catch пропишите логику, что если исключение было поймано, необходимо изменить цвет фона для текстового поля на красный (чтобы красный не выглядел вызывающим на форме, используйте цвет Color.LightPink).

11. В блоке try после присвоения напишите код, который устанавливает текстовому полю цвет на фона на белый Color.White. Таким образом, если значение было присвоено в свойство без ошибок, то цвет текстового поля будет нормальным. Если же во время присвоения произошла ошибка, то цвет текстового поля станет красноватым.

12. Запустите программу и убедитесь, что подсветка валидации действительно работает.

13. Теперь необходимо вывести пользователю сообщение с текстом ошибки, чтобы пользователь знал в чем ошибка и мог исправить данные. Если вы правильно выполнили задание с бизнес-логикой, тогда текст ошибки уже должен содержаться в исключении.

14. Добавьте в форму закрытое строковое поле ***_data***Error. Здесь вместо data надо подставить название поля, для которого мы сохраняем ошибку. Например, если мы валидируем текстовое поле с фамилией контакта, то новое строковое поле должно называться _surnameError.

15. Условимся, что если строковое поле содержит какой-то текст, значит, в текстовом поле есть некорректные данные. Если же строковое поле хранит пустую строку, значит, данные введены правильно. Реализуем данную логику.

16. В блоке try обработчика события TextChanged после присвоения данных в свойство добавим код, который присваивает в ***_data***Error пустую строку (если

присвоение прошло без ошибок, то надо сбросить любое сообщение об ошибке в строковом поле).

17. В блоке catch добавьте код, который присваивает в строковое поле ***_data***Error сообщение из исключения (exception.Message, где exception – переменная исключения).

18. Добавьте закрытый метод bool CheckFormOnErrors(). В методе сделайте проверку – если текстовое поле ***_data***Error не пустое, то нужно показать текстовое сообщение MessageBox с текстом из ***_data***Error, и вернуть false. Иначе вернуть true.

19. Добавьте обработчик события Click для кнопки OK. Добавьте в обработчик вызов метода CheckFormOnErrors(). Запустите программу и проверьте работу вывода сообщения по нажатию на кнопку OK.

20. Добавьте аналогичную валидацию и для других текстовых полей и элементов формы. Обратите внимание, что строковое поле с ошибкой, аналогичное ***_data***Error, надо добавлять для ***каждого*** элемента управления, который валидирует данные. При нажатии кнопки OK текстовое сообщение должно показывать сообщения ***всех*** неправильно введенных полей.

21. Создайте закрытый метод Update***Object***(). Метод должен обновлять данные в объекте ***_object*** данными с элементов пользовательского интерфейса. Добавьте вызов метода в обработчик кнопки OK.

22. Тщательно протестируйте работу окна. Проверьте исходный код, исправьте оформление кода, грамматические ошибки, добавьте xml-комментарии там, где их не хватает. Сделайте коммит в репозиторий.

23. В Visual Studio перейдите в ветку develop. Затем выполните слияние (merge) ветки features/6_add_secondary_form_logic в ветку develop. Теперь все изменения из ветки с добавленным решением должны переместиться в ветку develop. Убедитесь в этом с помощью GitHub.