Задание №7. Передача данных между окнами

Передача данных между формами

Передача данных между формами осуществляется также, как и передача данных между двумя обычными классами. Единственная разница заключается в том, что при передаче данных между формами необходимо обновить отображаемые на экране данные, если в результате передачи они поменялись.

Сначала рассмотрим пример передачи данных между двумя обычными классами. В этом примере у нас есть три класса: 1) Data – класс с условными данными, которые надо передавать между двумя классами; 2) Outer – внешний класс, который будет передавать данные во внутренний класс Inner и забирать их после обработки; 3) Inner – внутренний класс, который будет получать данные из внешнего класса Outer, обрабатывать их и возвращать обратно. Далее представлен код этих классов:

```
//Внешний класс
public class Outer
{
    //Данные, которые будут передаваться
    private Data _data = new Data();

    public void DoSomething()
    {
        //При необходимости инициализируем данные
        _data.Text = "Some text";
        _data.LastUpdate = DateTime.Now;

        var inner = new Inner(); //Создаем объект внутреннего класса
        inner.Data = _data; //Передаем ему данные из внешнего класса
        inner.UpdateData(); //Говорим внутреннему классу обработать данные
        _data = inner.Data; //После обработки забираем данные из внутреннего класса
    }
}

//Внутренний класс
public class Inner
{
    //Поле для временного хранения переданных данных
    private Data _data;
```

```
    //Свойство, через которое будут передаваться данные извне
    public Data Data
    {
        get
        {
            return _data;
        }
        set
        {
            _data = value;
        }
    }

    //Обрабатывает переданные в класс данные
    public void UpdateData()
    {
        _data.Text = "Text is updated";
        _data.LastUpdate = DateTime.Now;
    }
}

//Класс может содержать любые данные,
// в зависимости от целей вашей программы
public class Data
{
    public string Text;

    public DateTime LastUpdate;
}
```

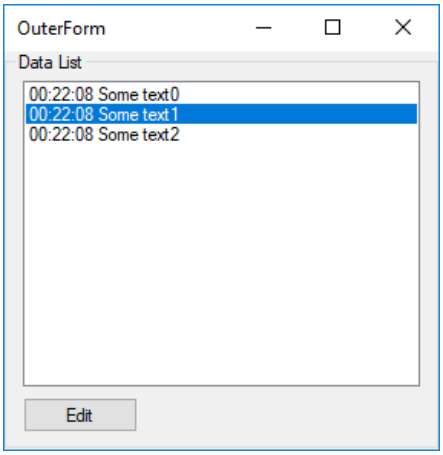
В качестве данных структура Data будет хранить строку Text и время последнего обновления строки LastUpdate. В вашей программе вместо класса Data может быть любой другой класс.

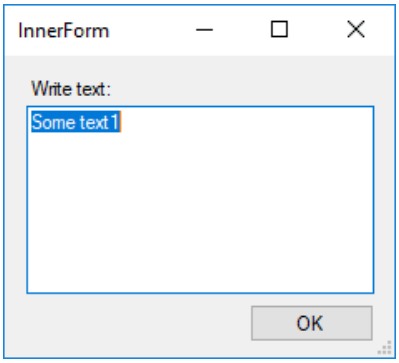
Для того, чтобы в класс Inner можно было передать данные, в нём должно быть свойство с открытым сеттером. Чтобы из класса Inner можно было забрать данные, в нём должно быть свойство с открытым геттером. В самом общем случае, входные и выходные данные могут быть представлены разными классами, например, Data1 и Data2.

Весь процесс обмена реализован в методе DoSomething() класса Outer. Класс Outer обращается к экземпляру класса Inner (который может создаваться внутри метода или быть агрегирован классом Outer), и через свойство Data передает собственный объект данных _data со строкой “Some text”. Теперь данные хранятся и в Outer, и в Inner.

Предположим, что внутри класса Inner эти данные поменялись. Например, был выполнен метод UpdateData(). Чтобы забрать данные из Inner, класс Outer обращается к геттеру _inner.Data и получает обновленный экземпляр данных со строкой “Text is updated” и новым значением времени в LastUpdate.

Теперь предположим, что вместо классов Outer и Inner нам необходимо передать данные между формами OuterForm и InnerForm. При этом, в внешней форме теперь будет хранится список данных Data, отображающихся в списке ListBox. А во внутренней форме InnerForm переданный экземпляр выводится на экран, где пользователь самостоятельно может ввести произвольный текст с помощью текстового поля:





Код OuterForm:

```
public partial class OuterForm : Form
{
    private List<Data> _data = new List<Data>();

    public OuterForm()
    {
        InitializeComponent();
        FillListBoxByTestData();
    }

    private void EditButton_Click(object sender, EventArgs e)
    {
        //Получаем текущую выбранную дату
        var selectedIndex = DataListBox.SelectedIndex;
        var selectedData = _data[selectedIndex];
```

```
        var inner = new InnerForm(); //Создаем форму
        inner.Data = selectedData; //Передаем форме данные
        inner.ShowDialog(); //Отображаем форму для редактирования
        var updatedData = inner.Data; //Забираем измененные данные

        //Осталось удалить старые данные по выбранному индексу
        // и заменить их на обновленные
        DataListBox.Items.RemoveAt(selectedIndex);
        _data.RemoveAt(selectedIndex);

        _data.Insert(selectedIndex, updatedData);
        var time = updatedData.LastUpdate.ToLongTimeString();
        var text = updatedData.Text;
        DataListBox.Items.Insert(selectedIndex, time + " " + text);
    }

    //Генерирует начальные тестовые данные для удобства тестирования
    // (не обязателен для примера)
    private void FillListBoxByTestData(int dataCount = 3)
    {
        for (int i = 0; i < 3; i++)
        {
            var data = new Data()
            {
                Text = "Some text" + i,
                LastUpdate = DateTime.Now
            };
            _data.Add(data);
            var time = data.LastUpdate.ToLongTimeString();
            var text = data.Text;
            DataListBox.Items.Add(time + " " + text);
        }
    }
}
```

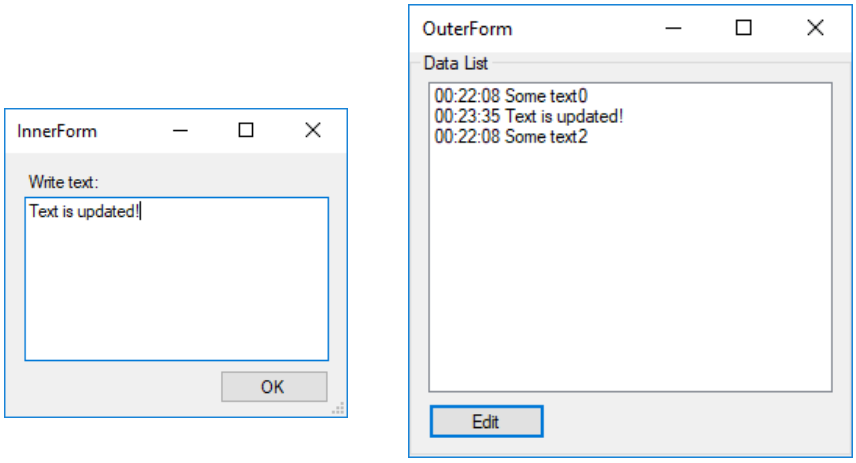
Если сравнить код OuterForm с классом Outer из предыдущего примера, то разница заключается лишь в обновлении интерфейса (пересоздаем элемент списка ListBox). Не считая метода FillListBoxByTestData(), написанного только для генерации тестовых данных, в код формы добавилось лишь 8 строчек кода. Исходный код InnerForm:

```
public partial class InnerForm : Form
{
    private Data _data;

    public Data Data
    {
        get
        {
            return _data;
        }
```

```
        set
        {
            _data = value;
            if (_data != null)
            {
                DataTextBox.Text = _data.Text;
            }
        }
    }

    public InnerForm()
    {
        InitializeComponent();
    }

    private void DataTextBox_TextChanged(object sender, EventArgs e)
    {
        _data.Text = DataTextBox.Text;
        _data.LastUpdate = DateTime.Now;
    }

    private void OkButton_Click(object sender, EventArgs e)
    {
        DialogResult = DialogResult.OK;
        this.Close();
    }
}
```

Опять же, реализация InnerForm мало отличается от класса Inner. В данном случае, изменение данных происходит не при вызове метода UpdateData(), а при действиях пользователя. Пользователь вводит новые данные в текстовое поле, после чего запускается обработчик события DataTextBox_TextChanged(), который и обновляет поля объекта _data. По нажатию на кнопку OK, внутренняя форма закрывается и управление возвращается в обработчик события EditButton_Click() формы OuterForm. Внешняя форма тут же забирает обновленные данные updatedData из уже закрытой внутренней формы, после чего обновляет список данных Data и пользовательский интерфейс. В данном случае, для обновления интерфейса, нам необходимо удалить строку в ListBox, которая была связана со старым объектом, и вставить на её место строку, сформированную из новых данных.

В результате работы получим:



То есть форма обновилась, и выделенная вторая строка теперь показывает новый введенный текст.

Для закрепления примера повторите реализацию этих форм. Для практики добавьте кнопку Cancel на InnerForm, отменяющую изменения текущего объекта. Подумайте самостоятельно, как её реализовать – задача не такая простая, как может показаться на первый взгляд.

На практике студенты часто используют другие способы передачи данных между формами. Сразу отметим, что эти способы неправильные:

1) Поместить текстовое поле DataTextBox внутренней формы под модификатор public, чтобы к нему можно было обратиться из внешней формы. Данный подход нарушает инкапсуляцию, что усложнит возможную модификацию внутренней формы.
2) Передать в качестве свойства внутренней формы не экземпляр Data, а экземпляр OuterForm, где список данных _data или ListBox помещены под модификатор доступа public. Этот способ еще более глупый, так как помимо нарушения инкапсуляции OuterForm, делает невозможным использование InnerForm без экземпляра OuterForm. Поддержка программы становится еще сложнее.
3) Передать экземпляр OuterForm в конструктор InnerForm. Аналогичен предыдущему способу.

Передать в InnerForm список всех данных _data вместо передачи только одного выбранного объекта. Данный подход опять же нарушает инкапсуляцию. Более того, подход заставляет передавать во внутреннюю форму больше данных, чем

ей нужно для работы. Передавать нужно только те данные, которые реально нужны для работы формы, передача же лишних данных может привести к их повреждению или потере.

Описание задания:

1. Создайте новую ветку в локальном репозитории под названием «features/7_make_forms_data_transfer». Перейдите в новую ветку.
1. Удалите из главного окна логику, выполняющую генерацию случайных объектов при нажатии на кнопку Add.

   Примечание: генерация случайных данных может помочь вам в будущей отладке приложения, поэтому можно пока не удалять данную функциональность – генерация работает быстрее, чем ручной ввод данных, и поэтому будущее тестирование сортировки, поиска и сохранение данных можно выполнять быстрее, если в программе временно останется добавление случайных данных. Вы можете добавить в меню главного окна пункт Edit -> Add Random ***Object*** и перенести логику добавления случайных данных в обработчик этого пункта меню.
2. Реализуйте взаимодействие между главным окном приложения и второстепенным. Теперь при нажатии на кнопку Add должно открываться новое второстепенное окно, в котором пользователь вводит данные об объекте (заметке, контакте или др.). При нажатии на кнопку OK второстепенное окно должно закрываться. После закрытия окна, главное окно должно забирать новый созданный объект данных из второстепенного окна, добавлять его в проект _project, а новый объект должен отобразиться в списке на главном окне.
3. Добавьте логику отмены для второстепенного окна. При нажатии на кнопку Cancel второстепенное окно должно закрываться, но никаких данных в проекте _project или на главном окне меняться не должно. Используйте для этого свойство DialogResult (см. Msdn).
4. Запустите приложение и убедитесь, что добавление новых данных работает корректно.
5. Реализуйте логику кнопки Edit главного окна. Для начала реализуйте закрытый метод Edit***Object***(int index). Метод должен принимать индекс заметки, которую нужно отдать во второстепенное окно для редактирования. Метод забирает объект из _project по указанному индексу и создаёт копию объекта (вызов метода Clone()). Копия объекта передается в экземпляр второстепенного окна, и окно отображается пользователю. После успешного редактирования заметки и по нажатию на кнопку OK, второстепенное окно закрывается, а измененный объект забирается из второстепенного окна. Важно, что измененный объект нужно поместить в _project по тому же самому индексу, где он находился

до редактирования (в отличие от добавления нового объекта, когда он добавлялся в конец списка). После обновления объекта в _project, его надо также обновить и в списке на главном окне, который видит пользователь.

6. Добавьте вызов метода Edit***Object***() в обработчики кнопки Edit и пункта главного меню Edit.
7. Запустите программу и убедитесь, что редактирование работает корректно.
8. Реализуйте корректную логику кнопки Cancel для второстепенного окна. Если пользователь во время редактирования нажмет кнопку Cancel, окно должно закрыться, но никаких изменений в данных _project и на главном окне произойти не должно.

   Возможность отмены действий обеспечивается с помощью свойства DialogResult и созданием копии исходного объекта. Работа с копией объекта во второстепенном окне гарантирует, что исходные данные не изменятся. Однако, предложенный вариант – не единственно правильный, и есть более оптимальная реализация, предполагающая создание копии внутри второстепенного окна.
9. Запустите программу и убедитесь, что программа работает корректно при добавлении, редактировании, удалении заметок, а также при отмене создания новой заметки и отмене редактирования выбранной заметки.
10. Убедитесь, что редактирование заметки действительно на срабатывает при нажатии на кнопку Cancel. Частая ошибка – когда пользователь нажал кнопку Cancel, но из-за недосмотра разработчика исходный объект всё равно был изменен.
11. В Visual Studio перейдите в ветку develop. Затем выполните слияние (merge) ветки features/7_make_forms_data_transfer в ветку develop. Теперь все изменения из ветки с добавленным решением должны переместиться в ветку develop. Убедитесь в этом с помощью GitHub.