Багатовіконні програми

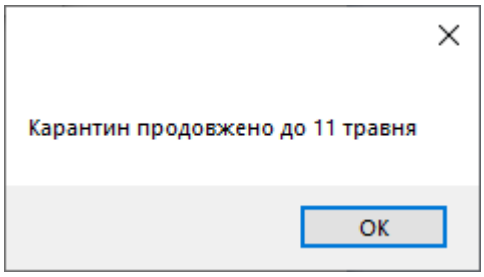
У багатьох випадках для взаємодії з користувачем не достатньо одного вікна. Наприклад, для коротких повідомлень користувачеві поверх головного вікна програми відкривають невелике вікно – модальну панель – з текстом і, можливо, зображенням. (Нове вікно можна відкрити в звичайному або модальному режимі. Модальне вікно не повертає фокус уведення головному, поки залишається відкритим.) Для отримання відповіді від користувача можна передбачити відповідні елементи керування в головному вікні, але, якщо відповідь потрібна лише іноді, то такі елементи загромаджуватимуть вікно. У багатьох випадках доцільніше відкрити з цією метою нове вікно діалогу. Усі стикалися з діалогами вибору імені файла для відкривання чи зберігання. Перелік можна продовжувати. Давайте знайомитися з конкретними прикладами.

Слайд 2

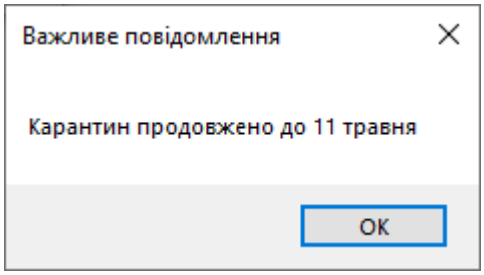
Статичний метод *Show* класу *MessageBox* відкриває просту інформаційну панель у модальному режимі. Є декілька перевантажених версій методу з різною кількістю параметрів. Параметри дають змогу налаштувати: текст повідомлення, заголовок панелі, відображення іконки, склад кнопок і вибір кнопки за замовчуванням.

Треба зазначити, що клас *MessageBox* не створює вікно, а звертається з запитом до ОС Windows по відповідний ресурс – модальну панель діалогу. Тому вигляд рамки панелі, іконок, кнопок і написів на них залежить від встановленої версії ОС.

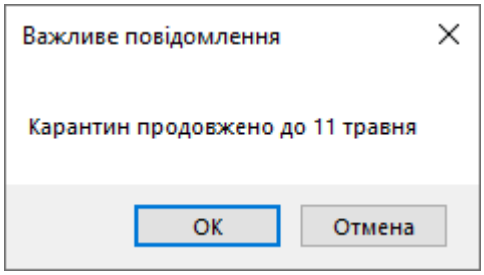
```
MessageBox.Show("Карантин продовжено до 11 травня");
```



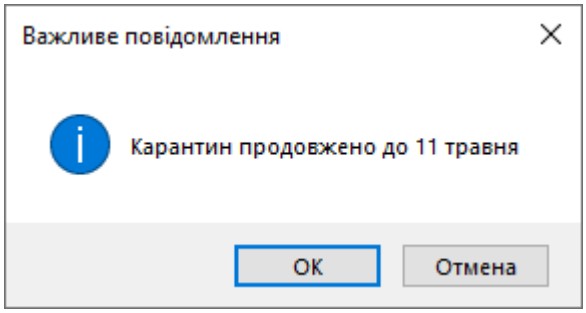

```
MessageBox.Show("Карантин продовжено до 11 травня", "Важливе повідомлення");
```



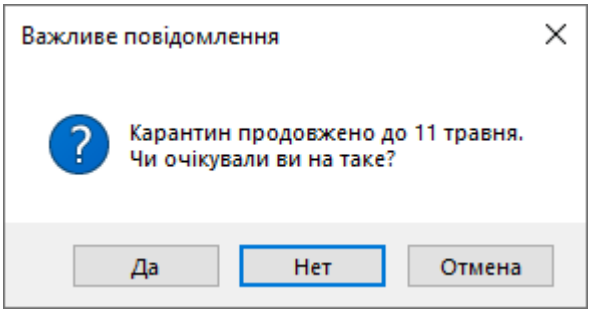

```
MessageBox.Show("Карантин продовжено до 11 травня", "Важливе повідомлення",
                MessageBoxButtons.OKCancel);
```



```
MessageBox.Show("Карантин продовжено до 11 травня", "Важливе повідомлення",
                MessageBoxButtons.OKCancel, MessageBoxIcon.Information); break;
```




```
MessageBox.Show("Карантин продовжено до 11 травня.\nЧи очікували ви на таке?",
                "Важливе повідомлення", MessageBoxButtons.YesNoCancel,
                MessageBoxIcon.Question, MessageBoxDefaultButton.Button2);
```



Метод *Show* повертає значення типу перелік *DialogResult*. Завдяки цьому програма може дізнатися, яку саме кнопку натис користувач. Можливість важлива, якщо від відповіді користувача залежить подальший хід виконання.

Слайд 3

Вибір кнопки панелі діалогу – це добре, але цього мало. Від користувача можна просити більше вхідних даних, наприклад, рядок. На жаль, бібліотека *WindowsForms* не містить готової компоненти для створення вікна, що повертає текстову відповідь користувача. Спроектуємо таке вікно власноруч.

1. Додати до проекту нову форму, налаштувати вигляд її рамки, наявність кнопок згортання, розгортання, початкове розташування на екрані (зазвичай по центру для панелей діалогу), розмір.
2. Додати до форми елементи керування. Щонайменше потрібні такі:
    a) напис, що міститиме запитання до користувача чи підказку про те, що саме програма хоче від нього отримати;
    b) рядок для введення відповіді;
    c) хоча б дві кнопки – для підтвердження та скасування введеної відповіді, обидві мали б закривати вікно діалогу

Слайд 4

3. Визначити конструктор форми, що задає початкові написи на всіх елементах керування чи на частині з них (наприклад, кнопки можуть мати незмінні написи).
4. Визначити відкриту властивість, що міститиме введений користувачем рядок. Саме так влаштовано стандартні діалоги: методи відкривання діалогу – *ShowDialog* (модальний) і *Show* (звичайний) – повертають значення типу *DialogResult*, тобто

ознаку того, яку саме кнопку натиснув користувач. А отримані від користувача дані екземпляр діалогу повертає через певну відкриту властивість. Дуже часто вона має тип *string*, хоча може мати довільний тип і повертати готовий сконструйований об'єкт цього типу.

5. Задати метод опрацювання натискання кнопки, що підтверджує відповідь користувача. Він повинен задавати значення властивості-результату, задавати результат діалогу (*this.DialogResult = DialogResult.OK;*) і закривати вікно діалогу. На опрацюванні кнопки скасування відповіді можна «зекономити»: достатньо задати у властивість форми *this.CancelButton = buttonCancel;* посилання на цю кнопку. Далі форма сама «знатиме», що потрібно закритися. До речі, задані кнопки підтвердження і скасування реагують на натискання [Enter] і [Esc] відповідно.

Слайд 5

Якийсь з елементів керування (кнопка, пункт меню) повинні створити екземпляр форми і відкрити його в модальному чи звичайному режимі. Далі потрібно проаналізувати результат діалогу: якщо він *DialogResult.OK*, то можна читати властивість-результат форми і працювати далі з отриманими від користувача даними. У протилежному випадку користувач скасував введення, нема чого опрацьовувати.

Після завершення діалогу потрібно звільнити ресурси ОС, які вікно отримало для функціонування, зокрема реєстрацію класу вікон, посилання на екземпляр вікна, на кнопки і рядок уведення. Для цього викликають метод *Dispode*.

У описаному сценарії екземпляр форми діалогу створюється кожного разу перед відкриванням діалогу і знищується після його завершення. Якщо такий діалог введення відбувається дуже часто, то багатократне створення-знищення сповільнюватиме виконання програми. Щоб уникнути таких затрат можна додати до класу форми посилання на екземпляр форми і створити його в конструкторі головної форми. Тоді метод запуску діалогу тільки відкриватиме його (без створення і знищення).

Модальний діалог – «одноразовий». Він дає змогу отримати від користувача одне значення. Якщо потрібно ввести серію значень, діалог можна відкривати у звичайному режимі. Влаштування «багаторазового» діалогу дещо складніше, адже кнопка підтвердження відповіді не повинна закривати вікно діалогу.

Як же головна форма довідається про те, що користувач щось увів, якщо вікно діалогу залишається відкритим? Перевірка *if (dlgRes == DialogResult.OK)* тут не підійде. Взаємодію між формами організовують за допомогою *подій*. Форма діалогу визначає подію «користувач увів дані», головна форма підписує якийсь окремий метод на опрацювання цієї події, тоді кнопка підтвердження відповіді задає значення властивості-результату й ініціює подію. Можна також передавати введене користувачем у аргументах події.

Слайд 6

Кожна сучасна програма має інформаційне вікно «Про програму», що повідомляє її призначення, версію, розробників тощо. Створення такого вікна у *WindowsForms* застосунку автоматизовано. Достатньо додати до проекту відповідну стандартну форму і налаштувати її властивості. Форма підтримує відображення текстових написів і графіки (статичної або gif-анімації). Версію програми, авторів, організацію форма автоматично зчитує з властивостей проекту.

Слайд 7

Дуже багато програм для обчислень використовують файли для зберігання вхідних даних і результатів. Редактори текстів чи табличні процесори призначені для опрацювання відповідних файлів. Усі такі програми потребують засобів для вибору файла для завантаження чи зберігання. *Діалоги вибору імені файла (файлів) надає операційна система*. Саме так. Файлові діалоги є об'єктами ОС. Саме від версії ОС залежить вигляд вікна діалогу вибору файла. Бібліотека *WindowsForms* надає *невізуальні* компоненти для налаштування діалогів на етапі проектування програми. Додана до проекту невізуальна компонента з'являється не на формі, а в лотку під нею. Властивості такої компоненти можна налаштовувати у вікні *Properties*, так само, як для візуальних компонент. Тільки результат налаштування не буде видно відразу. Наприклад, компонента *OpenFileDialog* використовує значення своїх властивостей для того, щоб передати їх системному діалогу вибору файла і відкрити його з налаштуваннями, заданими користувачем. Діалоги *WindowsForms* виступають посередниками між програмістом і відповідними об'єктами ОС.

Властивості *OpenFileDialog*:

- *Title* – заголовок вікна діалогу, бажано задавати завжди, інформувати користувача програми, для чого він вибирає файл;
- *DefaultExtension* – діалог може автоматично доповнювати ім'я файла заданим розширенням, якщо користувач його не введе;
- *Filter* – рядок спеціального вигляду "*пояснення_1*|*маска_1*|*пояснення_2*|*маска_2*| … |*пояснення_n*|*маска_n*", що налаштовує категорії файлів, які вибиратиме діалог для відображення, наприклад, "Text files|*.txt|All files|*.*";
- *CheckFileExist*, *CheckPathExist* – діалог може попереджати користувача, що він задав ім'я чи шлях, якого нема у файловій системі; та інші.

Слайд 8

Метод *ShowDialog* діалогової компоненти викликають у методі опрацювання події вибору пункту меню чи іншого елемента керування. І тоді легко повірити в магію компонент: чудесним чином відкривається справжній діалог, що надає доступ до дерева каталогів і їхнього вмісту. Розчарування настає так само швидко, як зачудування. ☺ Екземпляр *OpenFileDialog* насправді нічого не відкриває і не завантажує! Він *повертає рядок* – повне ім'я файла, який вибрав користувач (якщо вибрав). Цей рядок міститься у властивості *FileName*. Усю роботу з завантаження і відображення вмісту файла повинна виконати головна програма!

Простенький застосунок *WindowsFormsDialogs*, завантажений у папку лекції, демонструє типовий приклад використання файлових діалогів.

Слайд 9

Створити головне меню програми напрочуд легко: додайте до проекту компоненту *MenuStrip*. Вона з'являється у лотку форми, як невізуальна, але автоматично відображається одразу під заголовком вікна. Спочатку меню порожнє. Стандартними пунктами його можна наповнити автоматично, спеціальними – в режимі редагування. Все дуже швидко. Запустіть програму, і переконаєтеся, що щойно додане меню розгортається, є змога вибрати довільний його пункт мишкою чи комбінацією клавіш … – магія! Тільки ні один пункт не реагує на вибір. ☺ Після створення меню починається рутина: ручне програмування методів, що реагують на пункт меню.

Слайд 10

Панель інструментів – контейнер кнопок з піктограмами, що полегшують доступ до часто вживаних пунктів меню (дублюють їх). Автоматично можна додати стандартні кнопки. Роботу кнопок потрібно синхронізувати з відповідними пунктами меню.

Приклади побудови меню, використання вікна введення, вікна «Про програму» у застосунку *EditorSemiapplication* в папці лекції. Це не готова програма, а лише ескіз, який випробовує згадані засоби. Наприклад, тут без шкоди можна видалити-додати головне меню.

Слайди 11-13

Колись усі офісні програми мали багатодокументний інтерфейс: головне вікно редактора текстів Word, всередині якого відкривалися вікна документів. Так само з табличним процесором Excel. Сучасні програми значно оновили свій інтерфейс, а засоби для створення MDI залишилися. Можуть бути корисними, якщо ваша програма повинна опрацьовувати декілька файлів водночас – однотипних або й ні (дочірні вікна можуть бути різними).

Дочірні вікна додають до колекції дочірніх головного вікна. Головне вікно задають батьківським для дочірніх: батьківська форма завершує роботу дочірніх перед своїм завершенням.

Виринаючі підказки допомагають зорієнтуватися серед різноманітних елементів інтерфейсу. Їхньою появою керує окрема система аплікації, тому задавати доведеться в різних місцях різні властивості. Рядок меню чи панель інструментів є контейнерами, що відповідають за підказки вкладених компонент, тому налаштування двоетапне: властивість контейнера *ShowItemToolTip* встановити в *true*, тоді текст підказки до кожного інструмента задати в його властивості *ToolTipText*. Щоб задати підказки контролів, розташованих безпосередньо на формі, до форми додають невізуальну компоненту *ToolTip*, і в ній реєструють контроли разом з їхніми підказками.

Простий застосунок *IMDIApplication* демонструє окремі прийоми написання багатовіконних програм.

Слайд 14

Яка подія стається з застосунком, коли не стається ніяких подій? Подія *Idle*. Її опрацювання може виконувати корисну роботу з оновлення вигляду елементів керування, обчислювати статистику тощо, коли на це є час. Опрацювання *Idle* – проста альтернатива багатопотоковому програмування. Не повноцінна заміна, а простий спосіб виконання *короткотривалих* фонових завдань.

Єдина складність – на етапі проектування нема доступу до об’єкта *Application*, тому опрацювання події налаштовують програмно.

Слайд 15

Компоненту *DataGridView* можна використовувати для відображення структурованих об’єктів, колекцій об’єктів. Достатньо вказати колекцію джерелом даних, і компонента автоматично підлаштує свій вигляд та відобразить вміст колекції. Складом компоненти можна також керувати програмно.

Застосунок *WindowsFormsGrigApplication* демонструє різні варіанти використання *DataGridView*.

Ще один приклад багатовіконної програми – запозичений застосунок *NotepadCSharp*.