2017

# Методичні вказівки

**ДО ВИКОНАННЯ КУРСОВОЇ РОБОТИ З КУРСУ "ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ"**

СКЛАВ: ДОЦ., К.Т.Н. КОНОВАЛЕНКО І.В.

ТНТУ | Кафедра автоматизації технологічних процесів і виробництв
Розглянуто та затверджено на засіданні кафедри АВ,
протокол №3 від 19.09.2017

Методичні вказівки до виконання курсової роботи з курсу “Об'єктно-орієнтоване програмування”/ Уклад. Коноваленко І.В.– Тернопіль: ТНТУ, 2017.

Укладач: канд. техн. наук, доцент кафедри автоматизації технологічних процесів і виробництв Коноваленко І.В.

Методичні вказівки розглянуті та затверджені на засіданні кафедри автоматизації технологічних процесів і виробництв.

Протокол № 3 від 19 вересня 2017 р

# 1. Завдання на курсову роботу

Використовуючи принципи об'єктно-орієнтованого програмування, розробити застосунок Windows Forms, який дозволяє працювати з даними про об'єкти заданої предметної області. Для цього розробити клас, який описує об'єкт предметної області. Клас повинен містити не менше 10 властивостей різних типів і не менше двох методів (крім конструкторів), які забезпечать обробку даних об'єкта. Вид властивостей та методів слід вибрати самостійно шляхом аналізу заданої предметної області.

Застосунок, зокрема, повинен забезпечити:

- відображення даних про перелік об'єктів у табличному виді;
- ввід даних про новий об'єкт;
- зміну даних про попередньо введені об'єкти;
- видалення попередньо введених об'єктів по одному;
- повне очищення всього переліку за однією командою;
- відображення результату роботи обох методів класу;
- збереження даних про перелік об'єктів у двійковому форматі;
- читання попередньо збережених даних у двійковому форматі і відображення їх у табличному виді;
- експорт даних про перелік об'єктів у текстовий файл;
- імпорт даних з текстового файлу і відображення їх у табличному виді;
- сортування переліку даних за кількома найважливішими параметрами;
- пошук запису у переліку за одним з найважливіших параметрів;
- фільтрування даних у переліку за одним з найважливіших параметрів.

Для взаємодії з користувачем використати елементи керування "панель інструментів" (компонент ToolStrip середовища Microsoft Visual Studio), головне меню (MenuStrip) та "контекстне меню" (ContextMenuStrip).

Застосунок повинен містити не менше двох форм. Застосунок повинен забезпечити збереження базових налаштувань між сесіями роботи, до яких віднесемо: розміри (ширина, висота) головного вікна, його розміщення на екрані (відступ зліва, зверху), та порядок сортування даних.

Вид об'єкта (його предметну область) слід обрати згідно варіанту, або самостійно, узгодивши з викладачем. Можна також вибрати інше завдання, узгодивши його з викладачем.

Таблиця №1. Варіанти завдань на курсову роботу

| № | Об'єкт | № | Об'єкт |
|---|---|---|---|
| **1** | Комп'ютер | **16** | Годинник |
| **2** | Принтер | **17** | Книга |
| **3** | Автомобіль | **18** | Будівля |
| **4** | Літак | **19** | Планета |
| **5** | Фотоапарат | **20** | Футбольна команда |
| **6** | Жорсткий диск до ЕОМ | **21** | Рослина |
| **7** | Процесор | **22** | Тварина |
| **8** | Телефон | **23** | Університет |
| **9** | Телевізор | **24** | Фільм |
| **10** | Велосипед | **25** | Музичний виконавець |
| **11** | Планшет | **26** | |
| **12** | Ноутбук | **27** | |
| **13** | Абонент телефонної мережі | **28** | |
| **14** | Студент | **29** | |
| **15** | Персона | **30** | |

# 2. Приклад вирішення завдання

## 2.1. Уточнення завдання

Розглядатимемо предметну область: "місто". Проаналізувавши предметну область, вибираємо для об'єкта "місто" такі властивості:

1. Назва (тип даних – рядок).
2. Країна, у якій розташоване місто (тип даних – рядок).
3. Регіон, у якому розташоване місто (тип даних – рядок).
4. Кількість населення (тип даних – ціле число).
5. Річний бюджет (тип даних – число з плаваючою комою).
6. Площа (тип даних – число з плаваючою комою).
7. Географічна широта (тип даних – число з плаваючою комою).
8. Географічна довгота (тип даних – число з плаваючою комою).
9. Наявність порта (тип даних – логічний).
10. Наявність аеропорта (тип даних – логічний).

На основі даних про об'єкт "місто" слід розрахувати (сформувати):

1. Суму річного бюджету, яка припадає на одного мешканця (річний бюджет поділити на кількість мешканців).
2. Щільність населення у місті (кількість мешканців поділити на площу міста).
3. Узагальнений текстовий опис міста (назва + країна + регіон).

Застосунок має виконувати сортування списку з об'єктами "місто" за назвою, країною, регіоном, кількістю населення, площею, широтою та довготою.

Слід реалізувати можливість фільтрування даних у списку за параметром "площа міста". Користувач має задати мінімальне та максимальне значення площі, після чого застосунок відобразить у списку тільки ті записи, для яких об'єкти "місто" мають площу у вказаних межах.

Застосунок повинен забезпечити пошук запису за назвою міста.

## 2.2. Створення проекту застосунку

Створимо у середовищі Microsoft Visual Studio проект застосунку Windows Forms. Для цього слід вибрати команду меню File\New\Project, далі в розділі Templates\Visual C#\Windows вказати шаблон Windows Forms Application, вибрати розміщення проекту і задати назву для нього. У прикладі, який

розглядатиметься далі, назва проекту – CourseWork. Відповідно, так називатиметься і простір імен проекту.

Роботу над застосунком проводитимемо за таким планом:

1. Спроектуємо головне вікно застосунку. При цьому розмістимо на ньому елементи керування для здійснення всіх дій згідно завдання.
2. Запрограмуємо послідовно кожну операцію. При необхідності, розробимо додаткові форми для забезпечення виконання певних операцій.
3. Протестуємо застосунок.

## 2.3. Проектування головного вікна застосунку

Інтерфейс користувача визначає загальний вигляд вікон та розміщення і використання елементів керування, за допомогою яких користувач взаємодіє із застосунком. Ключовим елементом інтерфейсу нашого застосунку буде головне вікно, яке міститиме елементи керування для виконання всіх операцій згідно завдання.

### 2.3.1. Проектування структури головного вікна

Сформуємо головне вікно застосунку за типовою схемою, широко розповсюдженою серед різноманітного програмного забезпечення. Воно міститиме: смугу меню, панель інструментів, область з даними, статусний рядок (рис. 1).


Рис. 1. Структура головного вікна

Таку структуру використовують дуже часто, і вона забезпечує зручну взаємодію користувача із застосунком та швидкий доступ до його функціоналу.

Команди панелі інструментів та меню, як правило, дублюють. Меню містить зручний для сприйняття тестовий перелік команд, а панель інструментів надає швидкий доступ до них за допомогою набору кнопок (та інших елементів керування) з піктограмами. Область даних міститиме таблицю з введеними даними про об'єкти "місто". Для подання даних у табличному виді використаємо компонент DataGridView.

Панель інструментів забезпечуватиме засоби для виконання всіх операцій згідно завдання (додавання, редагування та видалення даних, їх збереження та читання, фільтрування, сортування і пошук). Панель інструментів сформуємо за допомогою компонента ToolStrip.

Меню міститиме доступ до всіх операцій, які реалізує панель інструментів, крім сортування та пошуку. Меню сформуємо за допомогою компонента MenuStrip.

Рядок стану міститиме додаткову інформацію: назву відкритого файлу, а також додаткові дані про вибраний в області даних об'єкт переліку. Рядок стану сформуємо за допомогою компонента StatusStrip.

Деякі операції, які виконуватиме застосунок, потребують додаткового введення даних. Так, для додавання (чи редагування) запису про об'єкт "місто" потрібно ввести значення його властивостей. Це завдання виконаємо за допомогою додаткового вікна, структура якого відповідатиме переліку властивостей об'єкта. Детальніше додаткові вікна розглянемо пізніше, під час реалізації відповідних операцій.

### 2.3.2. Проектування головного вікна

Кінцевий вигляд головного вікна застосунку, над яким будемо працювати, показано на рис. 2. Розглянемо проектування головного вікна детально.

Задамо назву для класу головного вікна (та файлів, які його описують). Для цього у Solution Explorer слід виділити елемент, який відповідає формі (як правило, на початковій стадії розробки проекту це Form1.cs) і з контекстного меню вибрати команду Rename. Задамо назву – FormMain.

Для головної форми слід задати такі властивості:

- StartPosition = CenterScreen;
- Text = "Курсова робота: міста"[1] (можна ввести свій опис застосунку);

[1] Значення всіх текстових властивостей у методичних вказівках взято в подвійні лапки (" "). Але при введенні цього значення у вікно Properties середовища Visual Studio лапки не потрібно вводити.

- Icon – вибрати довільну доречну піктограму (іконку) у форматі *.ico, яка відображуватиметься у верхньому лівому куті вікна. Саму піктограму знайти самостійно або вибрати з додаткових матеріалів до електронного навчального курсу "Об'єктно-орієнтоване програмування".

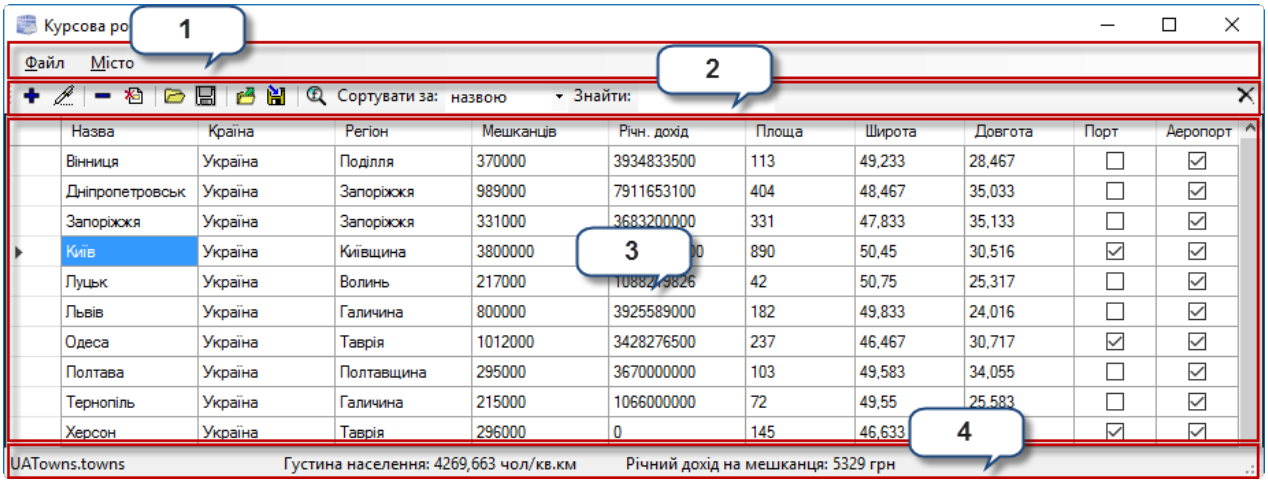


Рис. 2. Кінцевий вид головного вікна застосунку під час роботи (1 – область меню, 2 – панель інструментів, 3 – область даних, 4 – рядок стану

Розмістимо на формі компоненти MenuStrip, ToolStrip, StatusStrip та DataGridView. Задамо їм назви (Name у вікні Properties):

- компоненту MenuStrip – назву menuMain;
- компоненту ToolStrip – назву toolBar;
- компоненту StatusStrip – назву statusBar;
- компоненту DataGridView – назву gvTowns [2].

### 2.3.2.1. Розробка панелі інструментів

За допомогою кнопки з випадаючим списком компонента ToolStrip розмістимо на панель інструментів кнопки (Button), розділювачі (Separator), мітки (Label), комбінований список (ComboBox) та текстовий блок (TextBox) в такій послідовності (див. рис. 3, але початково на кнопках не буде зображень):

- дві кнопки,
- розділювач,
- дві кнопки,
- розділювач,
- дві кнопки,

[2] Префікс "gv" означає клас компонента – DataGridView.

- розділювач,
- дві кнопки,
- розділювач,
- кнопка,
- мітка,
- випадаючий список,
- мітка,
- текстове поле,
- кнопка.

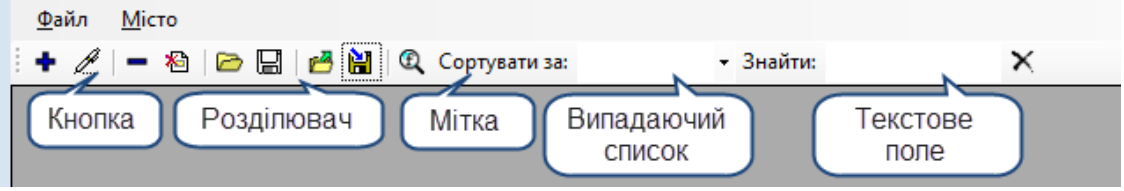


Рис. 3. Розташування елементів керування на панелі інструментів

Розглянемо призначення та налаштування кожної кнопки.

### *Кнопка "Додати запис про місто"*

Кнопка "Додати запис про місто" призначена для додання у список, який відображається в області даних, нового об'єкта "місто". Для кнопки слід задати такі властивості:

- Name = btnAdd [3];
- Image – вибрати доречну піктограму у форматі bmp, gif чи jpg. Саму піктограму знайти самостійно або вибрати з додаткових матеріалів до електронного навчального курсу "Об'єктно-орієнтоване програмування" (як задати піктограму, описано трохи нижче у цьому пункті)[4];
- ImageTransparentColor – вибрати колір прозорого фону зображення Image[5];
- Text = "Додати запис про місто";
- ToolTipText = "Додати запис про місто".

Щоб задати піктограму для кнопки на панелі інструментів, потрібно натиснути на кнопку "…" у властивості Image цієї кнопки. Відкриється вікно для вибору малюнка (рис. 4). Оскільки одні і ті ж піктограми використовуватимуться

[3] Префікс "btn" означає клас компонента – Button.
[4] Можна також завантажити з сайту Microsoft безкоштовну бібліотеку Visual Studio Image Library.
[5] Якщо зображення має одноманітний фон, який слід видалити і замінити фоном самої кнопки, то у цій властивості вказують колір фону. Колір можна вибрати один із стандартних (зі списку) або вказати довільний у форматі RGB.

і для кнопок, і для меню, то їх краще зберегти у ресурсному файлі проекту. Для цього слід активувати поле "Project resource file", після чого натиснути на кнопку "Import..." у цьому полі. Далі потрібно вибрати файл з доречною піктограмою на диску. Файл завантажиться у ресурси проекту і з'явиться у списку зліва. Натискання кнопки "ОК" приведе до встановлення вибраної піктограми для кнопки.

Якщо піктограму вже було завантажено раніше, її просто слід вибрати зі списку. У правій частині вікна (рис. 4) є поле для перегляду вибраної піктограми.

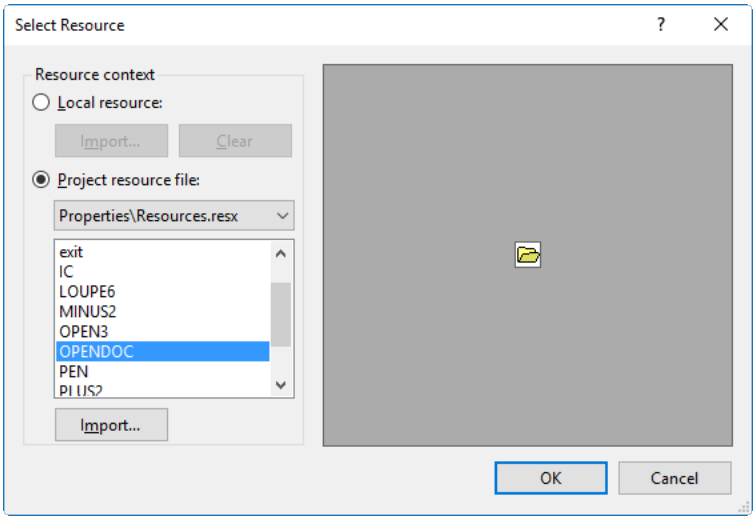


Рис. 4. Вибір піктограми

### *Кнопка "Редагувати запис"*

Кнопка "Редагувати запис" призначена для зміни даних вибраного елемента у списку об'єктів "місто". Для кнопки слід задати такі властивості:

- Name = btnEdit;
- Image – вибрати доречну піктограму у форматі bmp, gif чи jpg;
- ImageTransparentColor – вибрати колір прозорого фону зображення Image;
- Text = "Редагувати запис";
- ToolTipText = "Редагувати запис".

### *Перший розділювач*

З метою покращення візуального сприйняття елементів керування панелі інструментів використано розділювачі, які формують на панелі візуальні групи. Кожна така група міститиме функціонально близькі елементи керування. Перший розділювач відділяє (попередньо розглянуті) кнопки, які призначені для внесення даних, від інших елементів керування панелі. Для розділювача слід задати властивість:

- Name = tsSeparator1.

### *Кнопка "Видалити запис"*

Кнопка "Видалити запис" призначена для видалення зі списку вибраного об'єкта "місто". Для кнопки слід задати такі властивості:

- Name = btnDel;
- Image – вибрати доречну піктограму у форматі bmp, gif чи jpg;
- ImageTransparentColor – вибрати колір прозорого фону зображення Image;
- Text = "Видалити запис";
- ToolTipText = "Видалити запис".

### *Кнопка "Очистити дані"*

Кнопка "Очистити дані" призначена для повного очищення списку – видалення з нього всіх наявних записів про об'єкти "місто". Для кнопки слід задати такі властивості:

- Name = btnClear;
- Image – вибрати доречну піктограму у форматі bmp, gif чи jpg;
- ImageTransparentColor – вибрати колір прозорого фону зображення Image;
- Text = "Очистити дані";
- ToolTipText = "Очистити дані".

### *Другий розділювач*

Другий розділювач відділяє кнопки, які призначені для видалення даних, від наступних елементів керування панелі. Для розділювача слід задати властивість:

- Name = tsSeparator2.

### *Кнопка "Завантажити з бінарного файлу"*

Кнопка "Завантажити з бінарного файлу" призначена для читання попередньо збережених даних у бінарному форматі про перелік об'єктів "місто" та формування з них списку, який буде відображено в області даних вікна. Для кнопки слід задати такі властивості:

- Name = btnOpenFromBinary;
- Image – вибрати доречну піктограму у форматі bmp, gif чи jpg;
- ImageTransparentColor – вибрати колір прозорого фону зображення Image;
- Text = "Завантажити з бінарного файлу";
- ToolTipText = "Завантажити з бінарного файлу".

### *Кнопка "Зберегти у бінарному форматі"*

Кнопка "Зберегти у бінарному форматі" призначена для збереження даних про перелік об'єктів "місто" у бінарному форматі. Для кнопки слід задати такі властивості:

- Name = btnSaveAsBinary;
- Image – вибрати доречну піктограму у форматі bmp, gif чи jpg;
- ImageTransparentColor – вибрати колір прозорого фону зображення Image;
- Text = "Зберегти у бінарному форматі";
- ToolTipText = "Зберегти у бінарному форматі".

### *Третій розділювач*

Третій розділювач відділяє кнопки, які призначені для роботи з двійковими файлами, в яких зберігаються дані про об'єкти "місто". Для розділювача слід задати властивість:

- Name = tsSeparator3.

### *Кнопка "Завантажити з текстового файлу"*

Кнопка "Завантажити з текстового файлу" призначена для читання попередньо збережених даних про перелік об'єктів "місто" зі спеціально форматованого текстового файлу та формування з них списку, який буде відображено в області даних вікна. Для кнопки слід задати такі властивості:

- Name = btnOpenFromText;
- Image – вибрати доречну піктограму у форматі bmp, gif чи jpg;
- ImageTransparentColor – вибрати колір прозорого фону зображення Image;
- Text = "Завантажити з текстового файлу";
- ToolTipText = "Завантажити з текстового файлу".

### *Кнопка "Зберегти у текстовому форматі"*

Кнопка "Зберегти у текстовому форматі" призначена для збереження даних про перелік об'єктів "місто" у текстовому файлі. Для кнопки слід задати такі властивості:

- Name = btnSaveAsText;
- Image – вибрати доречну піктограму у форматі bmp, gif чи jpg;
- ImageTransparentColor – вибрати колір прозорого фону зображення Image;
- Text = "Зберегти у текстовому форматі";

- ToolTipText = "Зберегти у текстовому форматі".

### *Четвертий розділювач*

Четвертий розділювач відділяє кнопки, які призначені для роботи з текстовими файлами, у яких зберігаються дані про об'єкти "місто". Для розділювача слід задати властивість:

- Name = tsSeparator4.

### *Кнопка "Фільтрування даних"*

Кнопка "Фільтрування даних" призначена для виконання операції фільтрування даних у переліку об'єктів "місто". При її натисканні застосунок має відобразити вікно для введення критеріїв фільтрування (згідно уточненого завдання, це мінімальне та максимальне значення площі міста), після чого слід відобразити записи тільки про ті об'єкти "місто", які відповідають вказаним критеріям. Для кнопки слід задати такі властивості:

- Name = btnFilter;
- Image – вибрати доречну піктограму у форматі bmp, gif чи jpg;
- ImageTransparentColor – вибрати колір прозорого фону зображення Image;
- Text = "Фільтрування даних";
- ToolTipText = "Фільтрування даних".

### *Мітка "Сортувати за:"*

Мітка описує призначення наступного елемента керування – списку з параметрами сортування. Компоненту слід задати такі властивості:

- Name = tslSortBy [6];
- Text = "Сортувати за:".

### *Випадаючий список за параметрами сортування*

Список містить перелік параметрів сортування (згідно завдання). Вибір одного з параметрів сортування (наприклад, "за назвою", "за площею" тощо) має спричиняти сортування переліку даних про об'єкти "місто" у списку за заданим критерієм. Компоненту слід задати такі властивості:

- Name = tscbSortBy [7];
- DropDownStyle = DropDownList,

[6] Префікс "tsl" означає клас компонента – ToolStripLabel.
[7] Префікс "tscb" означає клас компонента – ToolStripComboBox.

- Items – слід ввести параметри сортування (приведені нижче фрази будуть продовженнями фрази "Сортувати за:" з попереднього компонента):
  - назвою
  - країною
  - регіоном
  - населенням
  - площею
  - широтою
  - довготою
- Size: Width = 90.

### *Мітка "Знайти:"*

Мітка описує призначення наступного елемента керування – текстового поля для пошуку. Компоненту слід задати такі властивості:

- Name = tslFind;
- Text = " Знайти:".

### *Текстовий блок для пошуку*

Текстовий блок призначений для введення назви міста, яке необхідно знайти у списку. При введенні символів застосунок має переходити до першого елемента у списку, назва якого починається із заданого рядка. Компоненту слід задати такі властивості:

- Name = tstbSearch [8];
- Size: Width = 100.

### *Кнопка "Вийти з програми"*

Кнопка "Вийти з програми" призначена для припинення виконання застосунку. Для кнопки слід задати такі властивості:

- Name = btnExit;
- Image – вибрати доречну піктограму у форматі bmp, gif чи jpg;
- ImageTransparentColor – вибрати колір прозорого фону зображення Image;
- Text = "Вийти з програми";
- ToolTipText = "Вийти з програми".

[8] Префікс "tstb" означає клас компонента – ToolStripTextBox.

Після розміщення та налаштування описаних вище елементів панель інструментів у дизайнері форм набуде вигляду відповідно до рис. 3 (але піктограми на кнопках можуть бути інші).

### 2.3.2.2. Розробка меню

Активувавши конструктор меню в дизайнері форм, слід створити меню зі структурою згідно рис. 5. На цьому малюнку показано текстові фрази, які представляють кожну команду меню. Вони зберігаються у властивостях Text кожного елемента. Якщо у меню між командами потрібно вставити розділювач (горизонтальну лінію), то як текст елемента слід ввести символ "-" (мінус). Знак "&" означає, що наступний після нього символ буде "гарячою клавішою" для цієї команди меню. Елемент меню викликатиметься при натисканні комбінації "Alt+Гаряча клавіша". Наприклад, команда "&Файл" (рис. 5) викликатиметься за допомогою комбінації клавіш Alt+Ф.

| **&Файл** | **&Місто** |
|---|---|
| Відкрити… | Додати… |
| Зберегти… | Змінити… |
| - | - |
| Імпортувати текст… | Видалити |
| Експортувати текст… | Очистити всі |
| - | - |
| Про програму | Фільтрувати… |
| - | |
| Вихід | |

} Верхній рівень меню

} Підменю

Рис. 5. Структура головного меню

Текст елементів меню, виклик яких приведе до появи інших діалогових вікон, прийнято закінчувати трикрапкою (наприклад, команда "Відкрити…" на рис. 5).

Назви (властивості Name) елементам меню слід задати згідно таблиці 2.

Задамо для команд меню піктограми, як і для кнопок, що виконують ті ж операції. Піктограму (вона буде розміщена зліва від тексту команди меню), задають у властивості Image. Для цього у конструкторі меню слід виділити потрібний елемент меню і натиснути на кнопку "…" у полі значення цієї властивості. З'явиться вікно, зображене на рис. 4. Так як ми вже завантажували у ресурсний файл проекту піктограми для кнопок, то у вікні слід активувати поле

"Project resource file", і вибрати зі списку потрібний малюнок (такий же, як і для відповідної кнопки).

Таблиця №2. Назви елементів головного меню

| Текст елемента меню | Назва (властивість Name) |
|---|---|
| **Файл** | **miFile** [9] |
| Відкрити… | miOpen |
| Зберегти… | miSave |
| Імпортувати текст… | miImport |
| Експортувати текст… | miExport |
| Про програму | miAbout |
| Вихід | miExit |

| Текст елемента меню | Назва (властивість Name) |
|---|---|
| **Місто** | **miTown** |
| Додати… | miAdd |
| Змінити… | miEdit |
| Видалити | miDelete |
| Очистити всі | miClear |
| Фільтрувати… | miFilter |
| | |

Для зручного доступу до елемента меню зручно використовувати "швидкі клавіші" – комбінацію клавіш (як правило, на основі клавіш Ctrl, Alt та Shift), за допомогою якої користувач може викликати відповідну команду меню. Щоб задати для елемента меню швидкі клавіші, слід скористатися його властивістю ShortcutKeys. Для цього слід натиснути на кнопку зі стрілкою в полі для значення властивості ShortcutKeys (рис. 6). У спливаючому вікні, яке з'явиться, слід маркером вибрати клавішу-модифікатор (Ctrl, Alt або Shift), а зі списку вибрати іншу клавішу клавіатури.

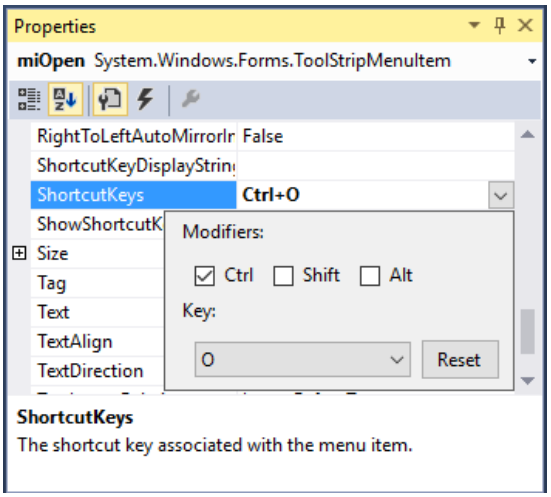


Рис. 6. Задання клавіш швидкого доступу для елемента меню

[9] Префікс "mi" означає клас компонента – MenuItem.

Задамо швидкі клавіші для таких команд:

| | |
|---|---|
| Відкрити… | Ctrl+O |
| Зберегти… | Ctrl+S |
| Вихід | Alt+X |

У результаті застосування всіх описаних дій щодо команд меню (задання тексту команд, піктограм та швидких клавіатурних комбінацій) конструктор меню у дизайнері форм виглядатиме відповідно до рис. 7 (хоча піктограми можуть бути іншими).

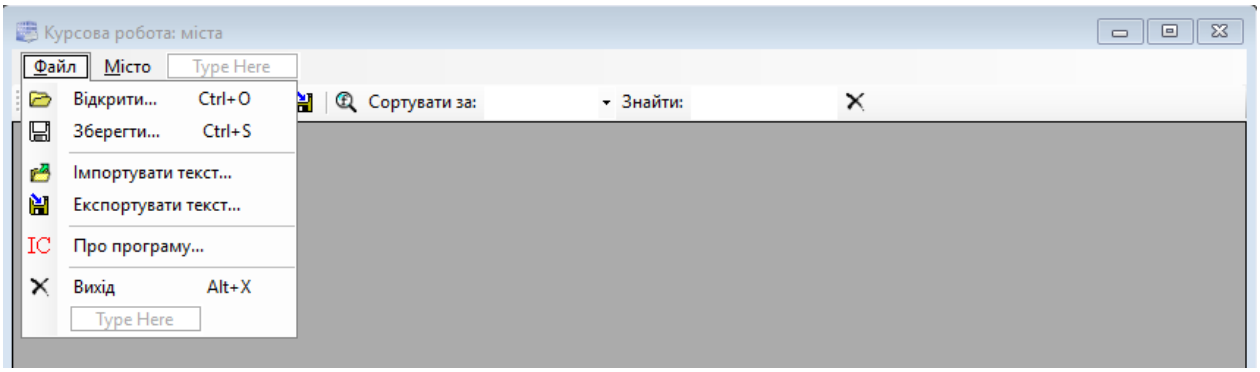


Рис. 7. Підменю "Файл" у конструкторі меню

### 2.3.2.3. Розробка рядка стану

Рядок стану відображатиме додаткову інформацію, а саме: назву відкритого файлу, густину населення вибраного у таблиці міста, і річний дохід на мешканця.

Щоб створити у рядку стану три поля, потрібно за допомогою кнопки компонента StatusStrip додати три мітки StatusLabel.

#### *Мітка для назви файлу*

Їй слід задати такі властивості:

- Name = tsslFileName[10];
- AutoSize = False;
- Size: Width = 200;
- TextAlign = MiddleLeft.

#### *Мітка для щільності населення*

[10] Префікс "tssl" вказує на тип компонента – ToolStripStatusLabel.

Їй слід задати такі властивості:

- Name = tsslPopDensity;
- AutoSize = False;
- Size: Width = 250;
- TextAlign = MiddleLeft.

### *Мітка для річного доходу на мешканця*

Їй слід задати такі властивості:

- Name = tsslYearIncomePerInhabitant;
- AutoSize = False;
- Size: Width = 200;
- TextAlign = MiddleLeft.

## 2.3.2.4. Налаштування компонента DataGridView та його джерела даних

Табличний компонент DataGridView (з назвою gvTowns) призначений для відображення даних про міста.

У ролі джерела даних для компонента DataGridView використаємо компонент BindingSource. Тому розмістимо на формі FormMain новий компонент – BindingSource – і задамо йому назву: Name = bindSrcTowns[11].

Розбиття стовпчиків даних для компонента та заповнення його даними виконуватиметься кодом під час виконання застосунку.

Для компонента таблиці DataGridView слід задати такі властивості:

- Name = gvTowns;
- AllowUserToAddRows = False;
- AllowUserToDeleteRows = False;
- DataSource = bindSrcTowns;
- Dock = Fill;
- MultiSelect = False;
- ReadOnly = True.

[11] Префікс "bindSrc" означає клас компонента – BindingSource.

### 2.3.2.5. Розробка контекстного меню

Контекстне меню викликається клацанням правої кнопки миші на компоненті. В нашому застосунку доцільно зробити контекстне меню для компонента gvTowns. Це меню дублюватиме кілька найвживаніших команд головного меню: "Додати…", "Змінити…", "Видалити" та "Вихід".

Щоб додати до проекту контекстне меню, слід розмістити на формі компонент ContextMenuStrip. Задамо для нього властивість Name = cmsMenuDataGrid [12].

Меню слід сформувати, ввівши елементи відповідно то таблиці №3. У цій же таблиці приведено назви (властивість Name) цих елементів меню.

Таблиця №3. Команди контекстного меню

| Текст елемента меню | Назва (властивість Name) |
|---|---|
| Додати… | mictAdd |
| Змінити… | mictEdit |
| - | |
| Видалити | mictDelete |
| - | |
| Вихід | mictExit |

Піктограми для елементів контекстного меню слід задати так само, як і для головного меню. При цьому команда, яка дублює певну команду головного меню, повинна мати таку ж піктограму.

Після того, як контекстне меню сформоване, його слід призначити для компонента gvTowns (адже різні компоненти можуть мати різні контекстні меню). Для цього у дизайнері форм слід вибрати компонент gvTowns і задати його властивість ContextMenuStrip = cmsMenuDataGrid (тут cmsMenuDataGrid – назва компонента контекстного меню).

---

На цьому проектування інтерфейсу головного вікна закінчено. В результаті всіх описаних вище дій над компонентами головної форми, вона (у дизайнері форм) має набути вигляду, подібного до рис. 8.

[12] Префікс "cms" означає клас компонента – ContextMenuStrip.

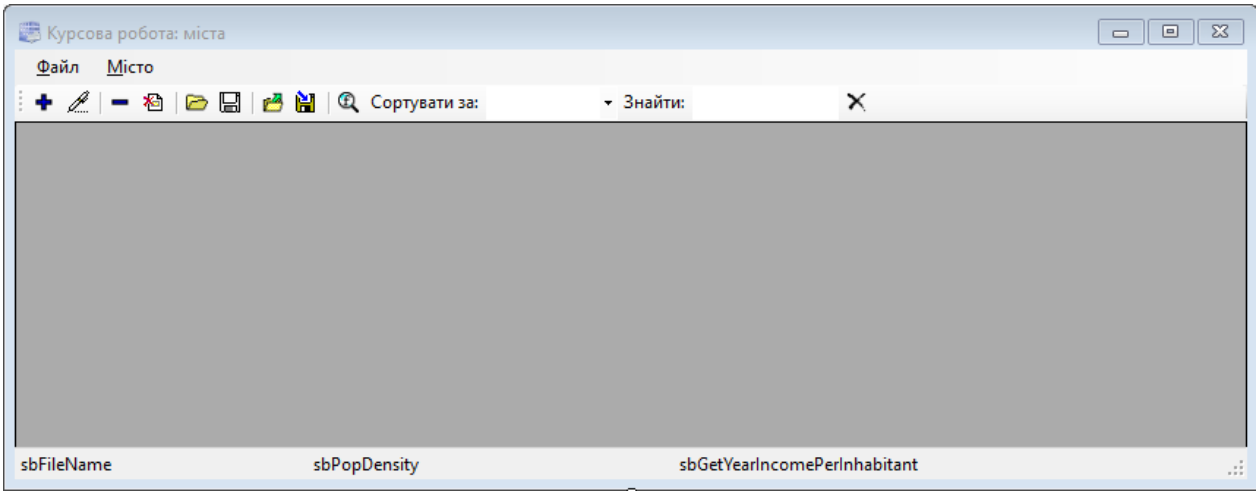


Рис. 8. Вид спроектованої головної форми у дизайнері форм

### 2.3.3. Проміжне тестування

Запустіть застосунок, натиснувши клавішу F5. Якщо попередні дії виконані вірно, має з'явитися головне вікно, вид якого відповідає рис. 8 (піктограми на кнопках можуть бути інші). Кнопки у вікні застосунку можна буде натискати, меню можна розгортати і вибирати команди, але, звісно, ніякі дії при цьому не виконуватимуться – адже нічого ще не запрограмовано. Щоб повернутися до середовища Visual Studio для подальшої роботи над проектом, слід закрити застосунок. Для цього слід скористатися кнопкою з хрестиком у правому верхньому кутку – команди для виходу з програми також не запрограмовані.

## 2.4. Програмування операцій

Після того, як дизайн головного вікна готовий, запрограмуємо послідовно кожну операцію згідно завдання. Але спочатку потрібно розробити клас, який інкапсулюватиме необхідні риси та поведінку об'єкта "місто". Цей клас буде основою, з якою працюватиме весь код, який стосується даних та їх обробки.

Далі реалізуємо найважливіші операції – для роботи з даними про об'єкт "місто". Це введення, редагування та видалення записів. Після цього запрограмуємо операції збереження (та читання) даних на диск. Далі реалізуємо операції, які маніпулюють даними: сортування, пошук та фільтрування. Останніми запрограмуємо додаткові дії: показ додаткової інформації у рядку стану та збереження налаштувань застосунку.

### 2.4.1. Проектування класу Town для опису об'єкта "місто"

#### 2.4.1.1. Опис членів класу Town

Назвемо клас, який інкапсулює об'єкт "місто" – Town. Повернувшись до уточненого завдання та проаналізувавши предметну область, складемо список членів класу, необхідних для реалізації поведінки об'єкта "місто" згідно завдання.

Для того, щоб містити необхідні (відповідно до завдання) дані про місто, клас Town повинен мати такі властивості:

| Назва | Опис назви | Тип | Опис типу |
|---|---|---|---|
| Name | Назва міста | string | Рядок |
| Country | Назва країни, у якій розташоване місто | string | Рядок |
| Region | Назва регіону, у якому розташоване місто | string | Рядок |
| Population | Кількість мешканців | int | Ціле число |
| YearIncome | Річний бюджет | double | Число з плаваючою комою |
| Area | Площа міста | double | Число з плаваючою комою |
| Latitude | Географічна широта | double | Число з плаваючою комою |
| Longitude | Географічна довгота | double | Число з плаваючою комою |
| HasPort | Наявність порта | bool | Логічний |
| HasAirport | Наявність аеропорта | bool | Логічний |

Для створення екземпляра об'єкта "місто" розробимо два конструктори: один з них створюватиме повністю "порожній" об'єкт з неініціалізованими даними, а другий – на основі переданих йому значень всіх вищезгаданих властивостей створюватиме екземпляр, який представлятиме реальний об'єкт "місто". Вид цих конструкторів (їх реалізацію детально розглянемо далі):

| Конструктор | Опис |
|---|---|
| `public Town()` | Створює "порожній" об'єкт |
| `public Town(string name, string country, string region, int population, double yearIncome, double area, double latitude, double longitude, bool hasPort, bool hasAirport)` | Створює об'єкт, який представляє конкретне місто |

Для реалізації необхідних згідно завдання обчислень над даними об'єкта "місто" та подання його розширеного текстового опису введемо у клас такі методи:

| Метод | Опис методу |
|---|---|
| `public string GeneralInfo()` | Формує розширений текстовий опис |
| `public double GetYearIncomePerInhabitant()` | Розраховує суму річного бюджету на одного мешканця |
| `public double PopulationDensity()` | Розраховує щільність населення |

Щоб забезпечити можливість сортування об'єктів "місто", клас Town має спадкувати інтерфейс IComparable. Це спричиняє необхідність реалізації класом ще одного методу, який належить цьому інтерфейсу – CompareTo():

| Метод | Опис |
|---|---|
| `public int CompareTo(object obj)` | Виконує порівняння двох об'єктів і вказує, який з них має передувати іншому |

### 2.4.1.2. Створення класу

Щоб додати у проект новий файл, де буде описано клас Town, слід з контекстного меню проекту в Solution Explorer вибрати команду Add\ New Item…, вказати шаблон Visual C# Items\ Class та задати назву файлу для опису класу – Town.cs (розширення можна не вказувати). Далі слід натиснути кнопку Add. Новий файл – Town.cs – з'явиться у списку Solution Explorer, і відкриється у редакторі коду.

У стандартній заготовці, яку Visual Studio створить для класу, слід у список базових класів додати назву інтерфейсу IComparable, і ввести опис всіх властивостей згідно лістингу 1. У лістингах код, який автоматично генерується середовищем, відзначено сірим. Цей код не повинен набиратися вручну. Якщо його немає, щось виконано не так. У цьому випадку слід перевірити попередні дії над проектуванням застосунку.

Всі властивості мають автоматичні підтримуючі поля (на це вказують порожні методи доступу).

Лістинг 1

```
public class Town : IComparable
{
    public string Name { get; set; }
    public string Country { get; set; }
    public string Region { get; set; }
    public int Population { get; set; }
    public double YearIncome { get; set; }
    public double Area { get; set; }
    public double Latitude { get; set; }
    public double Longitude { get; set; }
    public bool HasPort { get; set; }
    public bool HasAirport { get; set; }
```

Тепер введемо код конструкторів (лістинг 2). Перший конструктор "порожній". Його виклик приведе до створення екземпляра класу Town з неініціалізованими властивостями. Їх значення у подальшому можна задати. Другий конструктор отримує значення всіх властивостей, які містять дані про об’єкт "місто", і ініціалізує ними відповідні властивості. Таким чином, цей конструктор створює екземпляр, який відповідає конкретному місту.

Лістинг 2

```
public Town()
{

}

public Town(string name, string country, string region,
    int population, double yearIncome, double area,
    double latitude, double longitude, bool hasPort,
    bool hasAirport)
{
    Name = name;
    Country = country;
    Region = region;
    Population = population;
    YearIncome = yearIncome;
    Area = area;
    Latitude = latitude;
    Longitude = longitude;
    HasPort = hasPort;
    HasAirport = hasAirport;
}
```

Тепер додамо методи, які працюють з властивостями об’єкта (лістинг 3). Метод GeneralInfo() об’єднує значення текстових властивостей, формуючи

розширений опис міста за зразком: "Місто, країна, регіон" (наприклад, "Луцьк, Україна, Волинь").

Метод GetYearIncomePerInhabitant() обчислює долю міського бюджету, яка припадає на одного мешканця. Для цього сума міського бюджету ділиться на кількість мешканців. Але інколи кількість мешканців може бути не задана (тоді вона дорівнюватиме нулю). Щоб запобігти помилці ділення на нуль, метод виконує перевірку, чи властивість Population відмінна від нуля. Якщо Population дорівнюватиме нулю, метод поверне нуль. Метод GetYearIncomePerInhabitant() повертає дійсне число.

Лістинг 3

```
public string GeneralInfo()
{
    return Name + ", " + Country + ", " + Region;
}

public double GetYearIncomePerInhabitant()
{
    if (Population != 0) return YearIncome / Population;
    else return 0;
}

public double PopulationDensity()
{
    if (Area != 0) return Population / Area;
    else return 0;
}
```

Метод PopulationDensity() обчислює щільність (густину) мешканців, ділячи загальну кількість мешканців на площу міста. Як і в попередньому випадку, перевіряємо, чи знаменник виразу не дорівнює нулю. Метод PopulationDensity() повертає дійсне число.

### 2.4.1.3. Реалізація інтерфейсу IComparable

Інтерфейс IComparable, успадкований класом Town, має всього один метод – CompareTo(). Цей метод порівнює два екземпляри класу, і повертає -1, 0, або 1, вказуючи на порядковість їх взаємного розміщення. Метод CompareTo() використовується при сортуванні переліку об’єктів.

Згідно завдання, сортування об’єктів Town має проводитися за різними параметрами: за назвою, країною, регіоном, кількістю населення, площею, широтою та довготою. Таким чином, нам слід створити тип даних, яким можна буде задавати порядок сортування. Для цієї мети зручно використати перелік

(enum). Тому оголосимо у файлі Town.cs, в просторі імен проекту, але за межами оголошення класу Town, перелік згідно лістингу 4. Назви елементів цього переліку самозрозумілі (наприклад, SortByName означає "сортувати за властивістю Name – назвою міста тощо).

Лістинг 4

```
public enum TownsSortOrder
{
    SortByName = 1, SortByCountry = 2, SortByRegion = 3,
    SortByPopulation = 4, SortByArea = 5, SortByLatitude = 6,
    SortByLongitude = 7
};
```

А в клас Town додамо властивість SortOrder типу TownsSortOrder (лістинг 5), якою задаватиметься, які властивості об’єкта слід порівнювати у методі CompareTo().

Лістинг 5

```
    public TownsSortOrder SortOrder { get; set; }
```

Тепер реалізуємо метод CompareTo() інтерфейсу IComparable (лістинг 6). Його слід описати у класі Town.

В цей метод параметром obj типу object передається посилання на об’єкт, з яким слід порівняти даний об’єкт. Тому спочатку формуємо об’єкт t типу Town, "загорнувши" в нього переданий параметром об’єкт класу obj. Можна також сказати, що ми розглядаємо вміст пам’яті за посиланням obj як об’єкт класу Town.

Лістинг 6

```
public int CompareTo(object obj)
{
    Town t = obj as Town;
    switch(this.SortOrder)
    {
        case TownsSortOrder.SortByName:
            return string.Compare(this.Name, t.Name);
        case TownsSortOrder.SortByCountry:
            return string.Compare(this.Country, t.Country);
        case TownsSortOrder.SortByRegion:
            return string.Compare(this.Region, t.Region);
        case TownsSortOrder.SortByPopulation:
            return (this.Population > t.Population ? 1 :
              (this.Population < t.Population ? -1 : 0));
        case TownsSortOrder.SortByArea:
```

```
            return (this.Area > t.Area ? 1 : (this.Area < t.Area ?
              -1 : 0));
        case TownsSortOrder.SortByLatitude:
            return (this.Latitude > t.Latitude ? 1 :
              (this.Latitude < t.Latitude ? -1 : 0));
        case TownsSortOrder.SortByLongitude:
            return (this.Longitude > t.Longitude ? 1 :
              (this.Longitude < t.Longitude ? -1 : 0));
        default:
            return string.Compare(this.Name, t.Name);
    }

}
```

Далі за допомогою оператора вибору switch аналізуємо значення властивості SortOrder поточного екземпляра класу Town і, залежно від нього, порівнюємо відповідні властивості поточного і переданого об’єктів. При цьому для порівняння рядків (назва міста, країна, регіон) використано метод Compare класу string, а числові значення порівнюємо безпосередньо. Наприклад, при порівнянні площ, якщо площа (властивість Area) поточного об’єкта (this) більша за площу об’єкта, переданого методу (t), то метод повертає значення 1. Якщо площа поточного об’єкта менша, метод поверне -1. Якщо площі одинакові, метод поверне 0.

За замовчуванням (секція default, якщо значення властивості SortOrder не задане), сортування проводиться за назвою міста.

Код у файлі Town.cs використовує такі простори імен (цей код має бути з самого початку тексту):

```
using System;
```

---

Цілісно текст файлу Town.cs у складі проекту CourseWork можна завантажити з матеріалів електронного навчального курсу "Об’єктно-орієнтоване програмування в C#" (файл _CourseWork.zip).

### 2.4.2. Ініціалізація головного вікна

Під час запуску застосунку слід виконати ряд підготовчих дій, головна з яких – формування потрібного вигляду таблиці DataGridView. Цей компонент слід "відформатувати" так, щоб у ньому могла відображуватися інформація про об’єкти Town. Для цього у компоненті слід створити відповідні стовпчики.

Ініціалізація головного вікна виконується при його створенні, тому для цього зручно використати подію Load. Створимо за допомогою вікна Properties оброблювач події Load для головного вікна і запишемо туди код з лістингу 7.

Лістинг 7

```
tsslFileName.Text = tsslPopDensity.Text =
    tsslYearIncomePerInhabitant.Text = "";

gvTowns.AutoGenerateColumns = false;

DataGridViewColumn column = new DataGridViewTextBoxColumn();
column.DataPropertyName = "Name";
column.HeaderText = "Назва";
gvTowns.Columns.Add(column);

column = new DataGridViewTextBoxColumn();
column.DataPropertyName = "Country";
column.HeaderText = "Країна";
gvTowns.Columns.Add(column);

column = new DataGridViewTextBoxColumn();
column.DataPropertyName = "Region";
column.HeaderText = "Регіон";
gvTowns.Columns.Add(column);

column = new DataGridViewTextBoxColumn();
column.DataPropertyName = "Population";
column.HeaderText = "Мешканців";
gvTowns.Columns.Add(column);

column = new DataGridViewTextBoxColumn();
column.DataPropertyName = "YearIncome";
column.HeaderText = "Річн. дохід";
gvTowns.Columns.Add(column);

column = new DataGridViewTextBoxColumn();
column.DataPropertyName = "Area";
column.HeaderText = "Площа";
column.Width = 80;
gvTowns.Columns.Add(column);

column = new DataGridViewTextBoxColumn();
column.DataPropertyName = "Latitude";
column.HeaderText = "Широта";
column.Width = 80;
gvTowns.Columns.Add(column);

column = new DataGridViewTextBoxColumn();
column.DataPropertyName = "Longitude";
column.HeaderText = "Довгота";
column.Width = 80;
```

```
gvTowns.Columns.Add(column);

column = new DataGridViewCheckBoxColumn();
column.DataPropertyName = "HasPort";
column.HeaderText = "Порт";
column.Width = 60;
gvTowns.Columns.Add(column);

column = new DataGridViewCheckBoxColumn();
column.DataPropertyName = "HasAirport";
column.HeaderText = "Аеропорт";
column.Width = 60;
gvTowns.Columns.Add(column);

EventArgs args = new EventArgs();
OnResize(args);
```

Перша інструкція цього коду – "очищення" полів, які є складовими рядка стану. Це – назва відкритого файлу, густина населення та бюджет на одного мешканця. Згадані дані початково не визначені, тому відповідні поля очищуємо (якщо точніше, то – записуємо в них порожні рядки).

Далі налаштовуємо стовпчики компонента gvTowns для відображення даних з об'єктів Town. Спочатку властивості AutoGenerateColumns цього компонента присвоюємо значення false. Це означає, що налаштування стовпчиків виконуватиметься самостійно нашим застосунком, а не автоматично – засобами компонента.

Змінна column класу DataGridViewColumn представлятиме окремі стовпчики компонента gvTowns. Налаштування всіх стовпчиків здійснюється однаково. Спочатку створюємо новий екземпляр column класу DataGridViewColumn. Потім задаємо для нього властивості DataPropertyName (тут слід вказати назву властивості об'єкта класу Town, з якою буде пов'язаний стовпчик) та HeaderText (тут слід ввести назву стовпчика, яка відображуватиметься в його заголовку). Далі сформований таким чином стовпчик додаємо у колекцію стовпчиків Columns компонента gvTowns за допомогою метода Add().

Дві останні інструкції лістингу 7 імітують виникнення події Resize головного вікна. Оброблювач цієї події "переміщує" кнопку "Вийти з програми" на панелі інструментів головного вікна в кінець, до правого краю вікна. Це робиться для того, щоб відокремити цю кнопку від всіх інших елементів панелі. Код оброблювача події Resize головної форми приведено у лістингу 8. Лівий відступ

для кнопки btnExit формується на основі ширин 10 кнопок, 4 розділювачів, а також полів для сортування та пошуку.

Лістинг 8

```
private void FormMain_Resize(object sender, EventArgs e)
{
    int buttonsSize = 10 * btnAdd.Width + 4 * tsSeparator1.Width +
        tslSortBy.Width + tscbSortBy.Width + tslFind.Width +
        tstbSearch.Width + 30;
    btnExit.Margin = new Padding(Width - buttonsSize, 0, 0, 0);
}
```

Код у файлі FormMain.cs використовує такі простори імен (цей код має бути з самого початку):

```
using System;
using System.IO;
using System.Text;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Globalization;
```

### 2.4.2.1. Проміжне тестування

Запустіть застосунок, натиснувши клавішу F5. Якщо попередні дії виконані вірно, має з'явитися головне вікно, вид якого відповідає рис. 9. Таблиця повинна містити стовпчики з назвами. При зміні розмірів вікна кнопка для виходу має все одно залишатися справа. Щоб повернутися до середовища Visual Studio для подальшої роботи над проектом, слід закрити застосунок. Для цього слід скористатися кнопкою з хрестиком у правому верхньому кутку – адже команди для виходу з програми все ще не запрограмовані.

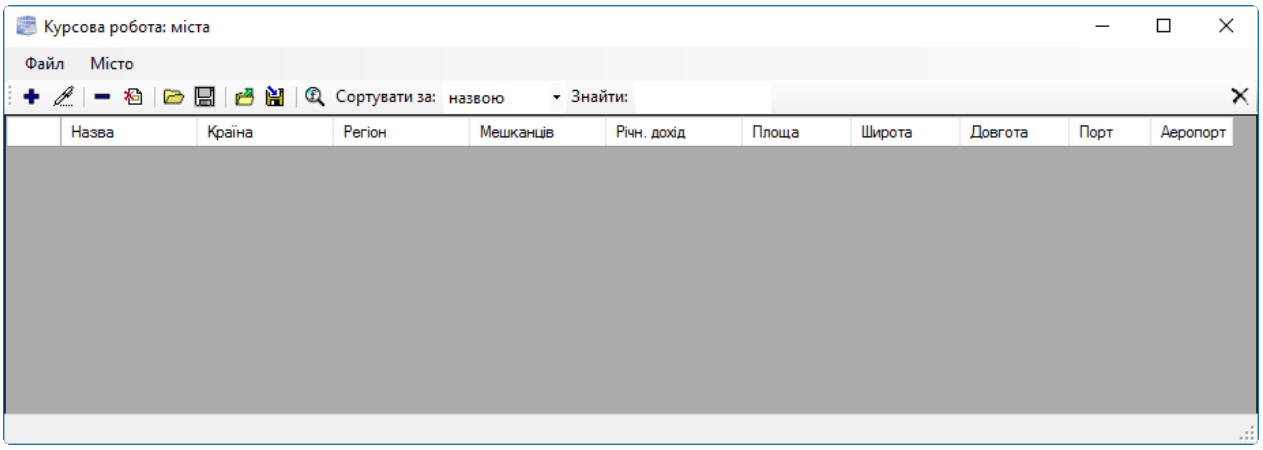


Рис. 9. Головне вікно без введених даних

### 2.4.3. Введення нових даних про місто

Для введення даних про новий об’єкт "місто" призначена кнопка btnAdd ("Додати запис про місто"). Оскільки об’єкт Town містить набір властивостей (назва, країна, регіон тощо), які слід задати при створенні нового запису, то потрібно передбачити засіб для їх введення.

Тому спроектуємо ще одну форму, яка матиме поля, що відповідають структурі властивостей для даних про місто.

#### 2.4.3.1. Проектування вікна для введення даних про місто

Щоб додати до проекту нову форму, слід викликати контекстне меню для проекту в Solution Explorer, вибрати команду Add\ New Item… і вказати шаблон Visual C# Items\ Windows Forms. Задамо назву для нового файлу (і класу) – FormTown. Після натискання кнопки Add новий елемент з вказаною назвою з’явиться у списку Solution Explorer.

Всі властивості об’єкта Town зручно розділити на дві групи. Перша – загальні дані (назви, мешканці, географічні координати, бюджет, площа), і друга – транспортні дані (наявність порта і аеропорта). Тому розмістимо на форму два компонента для груп – GroupBox. Компонент для загальних даних назвемо gbGeneral, а компонент для транспортних даних – gbTransport. Розмістимо їх один під другим. Для першого компонента задамо властивість Text = "Загальні дані", для другого Text = "Транспорт" (рис. 10).

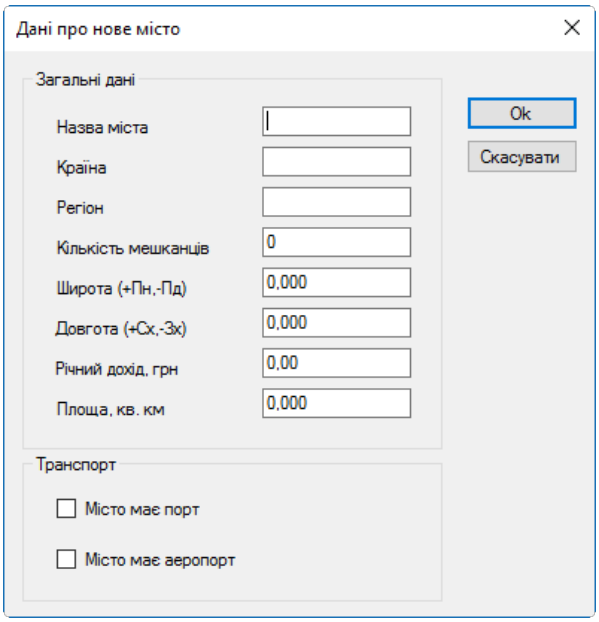


Рис. 10. Вікно для введення даних про місто

У першу групу ("Загальні дані") розмістимо 8 компонентів Label для написів і 8 компонентів TextBox для вводу даних. Розмістимо їх парами Label + TextBox, як показано на рис. 10. Для забезпечення рівномірності розташування компонентів (однакових проміжків між ними) потрібно виділити компоненти (у даному випадку всі компоненти TextBox, – для цього слід утримувати Shift і клацати на компонентах), після чого натиснути на кнопку "Make Vertical Spacing Equal" панелі інструментів Visual Studio. Далі потрібно розставити компоненти Label напроти відповідних компонентів TextBox.

Для компонентів Label задамо властивості Text відповідно до рис. 10.

Назви (властивість Name) для компонентів TextBox задамо такі:

| | |
|---|---|
| tbName | для назви міста |
| tbCountry | для назви країни |
| tbRegion | для назви регіону |
| tbPopulation | для кількості мешканців |
| tbLatitude | для широти |
| tbLongitude | для довготи |
| tbYearIncome | для річного бюджету |
| tbArea | для площі |

За назвами видно, до яких властивостей класу Town мають стосунок компоненти.

У другу групу ("Транспорт") розмістимо два компоненти CheckBox. Ці компоненти добре підходять для властивостей логічного типу, які можуть мати тільки два значення (true та false). Для першого компонента задамо:

- Name = chbHasPort[13];
- Text = "Місто має &порт".

Для другого:

- Name = chbHasAirport;
- Text = "Місто має &аеропорт".

Розмістимо на форму FormTown дві кнопки (рис. 10). Кнопка "Ok" призначена для підтвердження введених даних, а кнопка "Скасувати" – для скасування операції. Для першої кнопки задамо:

- Name = btnOk;

[13] Префікс "chb" означає тип компонента – CheckBox.

- Text = "&Ok".

Для другої:

- Name = btnCancel;
- DialogResult = Cancel;
- Text = "&Скасувати".

Для компонента форми задамо такі властивості:

- AcceptButton = btnOk;
- CancelButton = btnCancel;
- FormBorderStyle = FixedDialog;
- MaximizeBox = False;
- MinimizeBox = False;
- ShowInTaskBar = False;
- StartPosition = CenterParent;
- Text = "Дані про нове місто".

Після розміщення на формі всіх компонентів для них слід задати Tab-порядок, який визначає послідовність переходів між елементами керування при натисканні клавіші Tab. Для цього слід вибрати команду меню View\ Tab Order, і послідовно проклацати мишкою по всіх компонентах у дизайнері форм у потрібному порядку. Якщо Tab-порядок задано, слід ще раз вибрати команду меню View\ Tab Order.

Форму FormTown будемо використовувати для введення даних про нове місто, а також для редагування даних про вже існуюче місто. При створенні у цю форму передаватимемо екземпляр класу Town, у який записуватимемо введені користувачем дані. При редагуванні через цей екземпляр будемо передавати дані, які слід відобразити у вікні.

До класу форми додамо поле TheTown класу Town (лістинг 9)[14]:

Лістинг 9

```
public partial class FormTown : Form
{
    public Town TheTown;
```

[14] Нагадаємо ще раз: у лістингах виділений сірим код має бути згенерований середовищем Microsoft Visual Studio. Його не слід вводити вручну.

Конструктор класу перепишемо згідно лістингу 10. У конструктор передається посилання на екземпляр класу Town, яке запам’ятовується у щойно описаному полі TheTown.

Лістинг 10

```
public FormTown(Town t)
{
    TheTown = t;

    InitializeComponent();
}
```

При створенні вікна дані про переданий екземпляр слід відобразити у полях форми. Кожному полю присвоюється відповідне значення об’єкта TheTown. Тому створимо оброблювач події Load для форми і введемо туди код (лістинг 11).

Лістинг 11

```
private void FormTown_Load(object sender, EventArgs e)
{
    if (TheTown != null)
    {
        tbName.Text = TheTown.Name;
        tbCountry.Text = TheTown.Country;
        tbRegion.Text = TheTown.Region;
        tbPopulation.Text = TheTown.Population.ToString();
        tbYearIncome.Text = TheTown.YearIncome.ToString("0.00");
        tbArea.Text = TheTown.Area.ToString("0.000");
        tbLatitude.Text = TheTown.Latitude.ToString("0.000");
        tbLongitude.Text = TheTown.Longitude.ToString("0.000");
        chbHasPort.Checked = TheTown.HasPort;
        chbHasAirport.Checked = TheTown.HasAirport;
    }
}
```

При натисканні на кнопку "Ok" у вікні (рис. 10) слід перевірити введені дані і записати їх в об’єкт TheTown. Ці операції реалізовано в оброблювачі події Click кнопки btnOk. Його код приведено в лістингу 12. Перевірку правильності вводу числових значень виконуємо за допомогою методу TryParse() відповідного класу. Цей метод робить спробу перетворення рядка у числовий тип даних та повертає true у випадку успішного перетворення і false – при неможливості перетворення. В разі успіху результат повертається у вихідному параметрі методу.

Якщо введено помилкові дані, то за допомогою класу MessageBox показуємо користувачеві повідомлення про помилку, активуємо елемент керування з помилковими даними (за допомогою його методу Focus()), і припиняємо роботу методу.

Лістинг 12

```
private void btnOk_Click(object sender, EventArgs e)
{
    TheTown.Name = tbName.Text.Trim();
    TheTown.Country = tbCountry.Text.Trim();
    TheTown.Region = tbRegion.Text.Trim();

    int population;
    if (int.TryParse(tbPopulation.Text.Trim(), out population))
    {
        TheTown.Population = population;
    }
    else
    {
        MessageBox.Show("Неправильно введене число!", "",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        tbPopulation.Focus();
        return;
    }

    double d;
    if (double.TryParse(tbLatitude.Text.Trim(), out d))
    {
        TheTown.Latitude = d;
    }
    else
    {
        MessageBox.Show("Неправильно введене число!", "",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        tbLatitude.Focus();
        return;
    }
    if (double.TryParse(tbLongitude.Text.Trim(), out d))
    {
        TheTown.Longitude = d;
    }
    else
    {
        MessageBox.Show("Неправильно введене число!", "",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        tbLongitude.Focus();
        return;
    }
    if (double.TryParse(tbYearIncome.Text.Trim(), out d))
    {
        TheTown.YearIncome = d;
```

```
    }
    else
    {
        MessageBox.Show("Неправильно введене число!", "",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        tbYearIncome.Focus();
        return;
    }
    if (double.TryParse(tbArea.Text.Trim(), out d))
    {
        TheTown.Area = d;
    }
    else
    {
        MessageBox.Show("Неправильно введене число!", "",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        tbArea.Focus();
        return;
    }

    TheTown.HasPort = chbHasPort.Checked;
    TheTown.HasAirport = chbHasAirport.Checked;

    DialogResult = DialogResult.OK;
}
```

Якщо всі дані введено вірно, то (в останній інструкції методу btnOk_Click()) властивості DialogResult спроектованого нами діалогового вікна присвоюється значення DialogResult.OK. Присвоєння діалоговому вікну результату спричиняє його закривання. Результат DialogResult.OK означає підтвердження даних користувачем.

Коли користувач вирішить скасувати операцію, натиснувши кнопку "Скасувати", то діалоговому вікну присвоїмо результат DialogResult.Cancel. Це слід зробити в оброблювачі події Click кнопки btnCancel (лістинг 13).

Лістинг 13

```
private void btnCancel_Click(object sender, EventArgs e)
{
    DialogResult = DialogResult.Cancel;
}
```

Таким чином, дізнатися, чи користувач підтвердив введені дані, можна за допомогою результату діалогового вікна. Якщо він дорівнює DialogResult.OK, то користувач підтверджує введені дані і схвалює продовження операції

(додавання запису чи його редагування). Інший результат вказує на те, що операцію слід скасувати.

Код у файлі FormTown.cs використовує такі простори імен:

```
using System;
using System.Windows.Forms;
```

### 2.4.3.2. Програмування кнопки "Додати запис про місто"

Повернемося до головної форми – FormMain. Операцію додавання даних про нове місто реалізуємо при натисканні кнопки btnAdd на панелі інструментів головного вікна (в оброблювачі події Click кнопки). Загальна послідовність дій при цьому така (лістинг 14):

- створюємо "порожній" екземпляр town об’єкта класу Town;
- створюємо вікно ft класу FormTown (при цьому передаємо йому посилання на об’єкт town), показуємо вікно і чекаємо на його закриття;
- аналізуємо результат вікна: якщо він дорівнює DialogResult.OK (це означає, що в об’єкт town записано введені користувачем дані), додаємо до джерела даних bindSrcTowns новий екземпляр town;
- дані у таблиці DataGridView автоматично оновлюються і введені дані відображуються у вікні.

Лістинг 14

```
private void btnAdd_Click(object sender, EventArgs e)
{
    Town town = new Town();

    FormTown ft = new FormTown(town);
    if (ft.ShowDialog() == DialogResult.OK)
    {
        bindSrcTowns.Add(town);
    }
}
```

### 2.4.3.3. Проміжне тестування

Запустіть застосунок, натисніть на кнопку "Додати запис про місто", введіть дані і підтвердіть їх. У головному вікні у таблиці, яка представлена компонентом DataGridView, має з’явитися запис з введеними даними (рис. 11). Додайте таким чином кілька записів і перевірте, чи правильно відображуються дані.

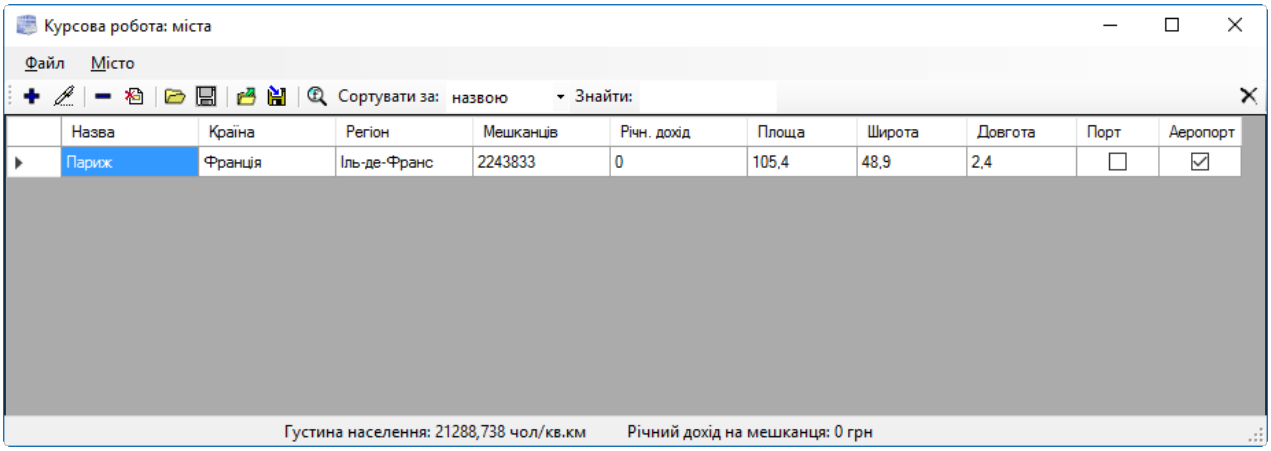


Рис. 11. Головне вікно з введеними даними про одне місто

### 2.4.4. Зміна даних про місто

Для зміни даних про місто призначена кнопка "Редагувати запис" на панелі інструментів головного вікна. Загальна послідовність дій при цьому така (лістинг 15):

- перевіряємо, чи є у джерелі даних якісь дані (якщо їх немає, то немає що редагувати). Якщо даних немає, припиняємо роботу методу;
- отримуємо посилання на поточний елемент множини даних bindSrcTowns і зберігаємо його у змінній town класу Town;
- створюємо вікно ft класу FormTown (при цьому передаємо йому посилання на об’єкт town), показуємо вікно (воно відображує всі поточні властивості town) і чекаємо на його закриття;
- аналізуємо результат вікна: якщо він дорівнює DialogResult.OK (це означає, що в об’єкт town записано відредаговані користувачем дані), змінюємо у джерелі даних bindSrcTowns поточний екземпляр списку на town;
- дані у таблиці DataGridView автоматично оновлюються і відредаговані дані відображуються у вікні.

Лістинг 15

```
private void btnEdit_Click(object sender, EventArgs e)
{
    if (bindSrcTowns.List.Count == 0) return;

    Town town = (Town)bindSrcTowns.List[bindSrcTowns.Position];

    FormTown ft = new FormTown(town);
    if (ft.ShowDialog() == DialogResult.OK)
    {
        bindSrcTowns.List[bindSrcTowns.Position] = town;
    }
}
```

#### 2.4.4.1. Проміжне тестування

Запустіть застосунок, введіть довільний запис про місто, потім відредагуйте його. Відредаговані дані мають відобразитися у таблиці. Перевірте, чи правильно змінюються дані.

### 2.4.5. Видалення запису про місто

Для видалення даних про місто призначена кнопка "Видалити запис" на панелі інструментів головного вікна. Загальна послідовність дій при цьому така (лістинг 16):

- перевіряємо, чи є у джерелі даних якісь дані (якщо їх немає, то немає що видаляти). Якщо даних немає, припиняємо роботу методу;
- перепитуємо, чи справді користувач хоче видалити запис;
- якщо так, видаляємо з джерела даних bindSrcTowns поточне значення;
- дані у таблиці DataGridView автоматично оновлюються і видалений запис зникає з таблиці.

Лістинг 16

```
private void btnDel_Click(object sender, EventArgs e)
{
    if (bindSrcTowns.List.Count == 0) return;

    if (MessageBox.Show("Видалити поточний запис?",
        "Видалення запису", MessageBoxButtons.OKCancel,
        MessageBoxIcon.Warning) == DialogResult.OK)
    {
        bindSrcTowns.RemoveCurrent();
    }
}
```

#### 2.4.5.1. Проміжне тестування

Запустіть застосунок, введіть кілька довільних записів про місто, потім виділіть середній запис і видаліть його. Він має зникнути з таблиці. Послідовно видаліть всі введені записи. Перевірте, чи правильно видаляються дані.

### 2.4.6. Очищення переліку міст

Для очищення переліку міст (видалення всіх даних) призначена кнопка "Очистити дані" на панелі інструментів головного вікна. Загальна послідовність дій при цьому така (лістинг 17):

- перевіряємо, чи є у джерелі даних якісь дані. Якщо даних немає, припиняємо роботу методу;
- перепитуємо, чи справді користувач хоче очистити таблицю;
- якщо так, очищуємо джерело даних bindSrcTowns;
- дані у таблиці DataGridView автоматично оновлюються, таблиця стає порожньою.

Лістинг 17

```
private void btnClear_Click(object sender, EventArgs e)
{
    if (bindSrcTowns.List.Count == 0) return;

    if (MessageBox.Show("Очистити таблицю?\n\n" +
        "Всі дані будуть втрачені", "Очищення даних",
        MessageBoxButtons.OKCancel,
        MessageBoxIcon.Question) == DialogResult.OK)
    {
        bindSrcTowns.Clear();
    }
}
```

### 2.4.6.1. Проміжне тестування

Запустіть застосунок, введіть кілька довільних записів про місто, потім очистіть таблицю. Всі записи мають зникнути з таблиці.

## 2.4.7. Збереження даних у файл

Збереження даних на диск має виконуватися при натисканні кнопки btnSaveAsBinary ("Зберегти у бінарному форматі"). При збереженні файлів використовують стандартне вікно для вибору розміщення та назви файлу. У Visual Studio воно представлене компонентом SaveFileDialog. Отже, розмістимо на форму FormMain такий компонент і задамо йому назву Name = saveFileDialog.

Загальний алгоритм збереження даних такий:

- перевіряємо, чи є у джерелі даних якісь дані. Якщо даних немає, припиняємо роботу методу;
- показуємо діалогове вікно SaveFileDialog для вибору розміщення та назви файлу. Якщо користувач підтвердив операцію збереження, то виконуємо подальші дії (якщо ні, припиняємо роботу методу);
- створюємо (перестворюємо) вказаний файл і відкриваємо його на запис;
- проходимо у циклі по всіх елементах списку джерела даних bindSrcTowns і записуємо всі поля поточного об’єкта Town у файл;
- закриваємо файл.

Цей алгоритм реалізовано в лістингу 18. Тут приведено оброблювач події Click для кнопки btnSaveAsBinary. Спочатку для діалогового вікна SaveFileDialog встановлюємо властивості Filter, Title та InitialDirectory. Властивість Filter задає тип файлів, які відображуються у вікні. В нашому випадку тип (розширення) файлів – .towns. Щоб діалогове вікно відкривалося у папці, звідки запущено застосунок, його властивості InitialDirectory задамо відповідне значення (Application.StartupPath).

Лістинг 18

```
private void btnSaveAsBinary_Click(object sender, EventArgs e)
{
    if (bindSrcTowns.List.Count == 0) return;

    saveFileDialog.Filter =
        "Файли даних (*.towns)|*.towns|All files (*.*)|*.*";
    saveFileDialog.Title = "Зберегти дані у бінарному форматі";
    saveFileDialog.InitialDirectory = Application.StartupPath;

    BinaryWriter bw;

    if (saveFileDialog.ShowDialog() == DialogResult.OK)
    {
        bw = new BinaryWriter(saveFileDialog.OpenFile());
        try
        {
            foreach (Town town in bindSrcTowns.List)
            {
                bw.Write(town.Name);
                bw.Write(town.Country);
                bw.Write(town.Region);
                bw.Write(town.Population);
                bw.Write(town.YearIncome);
                bw.Write(town.Area);
                bw.Write(town.Latitude);
                bw.Write(town.Longitude);
                bw.Write(town.HasPort);
                bw.Write(town.HasAirport);
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Сталась помилка: \n{0}", ex.Message,
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        finally
        {
            bw.Close();
        }
    }
}
```

Для запису бінарних даних у файл використаємо клас BinaryWriter.

З метою обробки можливих виняткових ситуацій використано блок try…catch. У випадку їх виникнення користувач отримає повідомлення з описом об’єкта (ex.Message) виняткової ситуації. Щоб файл, з яким проводиться робота, гарантовано був закритий, операція закриття виконується у блоці finally.

#### 2.4.7.1. Проміжне тестування

Запустіть застосунок, введіть один запис про місто і збережіть його. Перевірте поведінку застосунку: діалогове вікно для вибору файлу має відкриватися у папці застосунку, після збереження на диску має з’явитися файл із вказаною при збереженні назвою та розширенням town.

### 2.4.8. Читання збережених даних

Читання попередньо збережених на диску даних і відображення їх у таблиці головного вікна має виконуватися при натисканні кнопки btnOpenFromBinary ("Завантажити з бінарного файлу"). Тому відповідні дії слід програмувати в оброблювачі події Click цієї кнопки.

При відкриванні файлів використовують стандартне вікно для вибору файлу. У Visual Studio воно представлене компонентом OpenFileDialog. Отже, розмістимо на форму FormMain такий компонент і задамо йому назву Name = openFileDialog.

Загальний алгоритм читання даних такий:

- показуємо діалогове вікно OpenFileDialog для вибору файлу. Якщо користувач підтвердив операцію відкривання, то виконуємо подальші дії (якщо ні, припиняємо роботу методу);
- очищуємо джерело даних bindSrcTowns;
- відкриваємо вказаний файл на читання;
- доки не досягнуто кінця файлу, послідовно зчитуємо дані для кожного міста, формуємо на основі них об’єкт Town і додаємо його у джерело даних bindSrcTowns. Дані автоматично відображуються у таблиці головного вікна;
- закриваємо файл.

Цей алгоритм реалізовано в лістингу 19. Для читання бінарних даних використано клас BinaryReader.

Лістинг 19

```
private void btnOpenFromBinary_Click(object sender, EventArgs e)
{
    openFileDialog.Filter =
        "Файли даних (*.towns)|*.towns|All files (*.*)|*.*";
    openFileDialog.Title = "Прочитати дані у бінарному форматі";
    openFileDialog.InitialDirectory = Application.StartupPath;

    BinaryReader br;

    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        bindSrcTowns.Clear();

        br = new BinaryReader(openFileDialog.OpenFile());

        try
        {
            Town town;

            while (br.BaseStream.Position < br.BaseStream.Length)
            {
                town = new Town();
                for (int i = 1; i <= 10; i++)
                {
                    switch (i)
                    {
                        case 1:
                            town.Name = br.ReadString();
                            break;
                        case 2:
                            town.Country = br.ReadString();
                            break;
                        case 3:
                            town.Region = br.ReadString();
                            break;
                        case 4:
                            town.Population = br.ReadInt32();
                            break;
                        case 5:
                            town.YearIncome = br.ReadDouble();
                            break;
                        case 6:
                            town.Area = br.ReadDouble();
                            break;
                        case 7:
                            town.Latitude = br.ReadDouble();
                            break;
                        case 8:
                            town.Longitude = br.ReadDouble();
                            break;
                        case 9:
```

```
                            town.HasPort = br.ReadBoolean();
                            break;
                        case 10:
                            town.HasAirport = br.ReadBoolean();
                            break;
                    }
                }
                bindSrcTowns.Add(town);
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Сталась помилка: \n{0}", ex.Message,
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        finally
        {
            br.Close();
        }

        tsslFileName.Text = openFileDialog.SafeFileName;

        //tscbSortBy_SelectedIndexChanged(null, null);
    }
}
```

Після завантаження даних з файлу у поле tsslFileName статусного рядка (це компонент мітки – Label) записуємо назву файлу, з якого читали дані. Далі через виклик методу tscbSortBy_SelectedIndexChanged запуститься механізм сортування даних. Оскільки сортування ще не реалізоване, то відповідний (останній) рядок коду закоментовано. В подальшому, коли метод tscbSortBy_SelectedIndexChanged буде реалізовано, символи коментаря("//") видалимо.

### 2.4.8.1. Проміжне тестування

Запустіть застосунок, натисніть на кнопку "Завантажити з бінарного файлу" і відкрийте попередньо збережені дані. Перевірте поведінку застосунку: діалогове вікно для вибору файлу має відкриватися у папці застосунку, після читання даних вони мають відобразитися у таблиці головного вікна.

## 2.4.9. Експорт даних у текстовий файл

Експорт (запис) даних у текстовий файл виконується за допомогою кнопки btnSaveAsText ("Зберегти у текстовому форматі") на панелі інструментів головного вікна застосунку.

Загальна послідовність дій при цьому така ж, як і при збереженні даних у бінарному форматі, але робота ведеться з текстовим файлом (лістинг 20). Для запису в текстовий файл використано клас StreamWriter.

Лістинг 20

```
private void btnSaveAsText_Click(object sender, EventArgs e)
{
    if (bindSrcTowns.List.Count == 0) return;

    saveFileDialog.Filter =
        "Текстові файли (*.txt)|*.txt|All files (*.*)|*.*";
    saveFileDialog.Title = "Зберегти дані у текстовому форматі";
    saveFileDialog.InitialDirectory = Application.StartupPath;

    StreamWriter sw;

    if (saveFileDialog.ShowDialog() == DialogResult.OK)
    {
        sw = new StreamWriter(saveFileDialog.FileName,
            false, Encoding.UTF8);
        try
        {
            foreach (Town town in bindSrcTowns.List)
            {
                sw.Write(town.Name + "\t" + town.Country + "\t" +
                  town.Region + "\t" +
                  town.Population + "\t" +
                  town.YearIncome + "\t" + town.Area + "\t" +
                  town.Latitude + "\t" + town.Longitude + "\t" +
                  town.HasPort + "\t" + town.HasAirport + "\t\n");
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Сталась помилка: \n{0}", ex.Message,
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        finally
        {
            sw.Close();
        }
    }
}
```

### 2.4.9.1. Проміжне тестування

Запустіть застосунок і відкрийте попередньо збережені дані. Тепер спробуйте зберегти дані у текстовому форматі. Для цього натисніть кнопку "Зберегти у текстовому форматі", введіть назву файлу і збережіть дані. На диску

має з’явитися файл із заданою назвою і розширенням txt. Спробуйте відкрити цей файл за допомогою стандартного застосунку Windows "Блокнот" (Notepad). У "Блокноті" має бути повний текстовий опис даних, які відображуються в головному вікні.

### 2.4.10. Імпорт даних з текстового файлу

Імпорт (читання) даних з текстового файлу виконується за допомогою кнопки btnOpenFromText ("Завантажити з текстового файлу") на панелі інструментів головного вікна застосунку.

Загальна послідовність дій при цьому така ж, як і при читанні даних у бінарному форматі, але робота ведеться з текстовим файлом (лістинг 21). Для запису в текстовий файл використано клас StreamReader.

Лістинг 21

```
private void btnOpenFromText_Click(object sender, EventArgs e)
{
    openFileDialog.Filter =
        "Текстові файли (*.txt)|*.txt|All files (*.*)|*.*";
    openFileDialog.Title = "Прочитати дані у текстовому форматі";
    openFileDialog.InitialDirectory = Application.StartupPath;

    StreamReader sr;

    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        bindSrcTowns.Clear();

        sr = new StreamReader(openFileDialog.FileName,
            Encoding.UTF8);
        string s;
        try
        {
            while ((s = sr.ReadLine()) != null)
            {
                string[] split = s.Split('\t');
                Town town = new Town(split[0], split[1], split[2],
                  int.Parse(split[3]), double.Parse(split[4]),
                  double.Parse(split[5]), double.Parse(split[6]),
                  double.Parse(split[7]), bool.Parse(split[8]),
                  bool.Parse(split[9]));
                bindSrcTowns.Add(town);
            }

        }
        catch (Exception ex)
        {
            MessageBox.Show("Сталась помилка: \n{0}", ex.Message,
```

```
                MessageBoxButtons.OK, MessageBoxIcon.Error)
        }
        finally
        {
            sr.Close();
        }

        tsslFileName.Text = openFileDialog.SafeFileName;
    }
}
```

### 2.4.10.1. Проміжне тестування

Запустіть застосунок і відкрийте попередньо збережені у текстовому форматі дані. Перевірте, чи дані правильно зчитано і відображено.

## 2.4.11. Фільтрування даних

Фільтрування даних має відбуватися при натисканні на кнопку "Фільтрування даних" на панелі інструментів головного вікна. Операція фільтрування полягає у відкиданні всіх даних, які не задовольняють потрібним критеріям. У нашому випадку таким критерієм є площа міста. Користувач повинен ввести мінімальне та максимальне значення площі, після чого застосунок має відобразити тільки ті міста, для яких площа знаходиться в заданих межах. Таким чином, потрібно спроектувати засіб, який дозволить користувачеві ввести потрібні значення площ.

Для вирішення цього завдання спроектуємо ще одне просте вікно.

### 2.4.11.1. Проектування вікна для введення параметрів фільтра

Щоб додати до проекту нову форму, слід викликати контекстне меню для проекту в Solution Explorer, вибрати команду Add\ New Item… і вказати шаблон Visual C# Items\ Windows Forms. Задамо назву для нового файлу (і класу) – FormFilter. Після натискання кнопки Add новий елемент з вказаною назвою з’явиться у списку Solution Explorer.

Загальний вигляд вікна для введення параметрів фільтра показано на рис. 12.

Перейдемо у конструктор форм з новою формою. Розмістимо на формі компонент групування GroupBox, назвемо його (властивість Name) gbArea, і задамо властивість Text = "Площа".

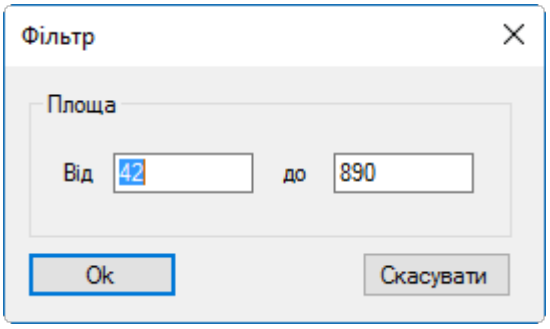


Рис. 12. Вікно для введення параметрів фільтра

На компонент групи помістимо два компоненти Label та два компоненти TextBox, у порядку: Label – TextBox – Label – TextBox (рис. 12). Текст для міток слід ввести згідно поданого малюнка, а назви компонентам TextBox слід задати відповідно tbAreaMin та tbAreaMax.

Нижче розмістимо ще дві кнопки (компоненти Button). Кнопка "Ok" призначена для підтвердження введених даних, а кнопка "Скасувати" – для скасування операції. Для першої кнопки задамо:

- Name = btnOk;
- Text = "&Ok".

Для другої:

- Name = btnCancel;
- DialogResult = Cancel;
- Text = "&Скасувати".

Для компонента форми задамо такі властивості:

- AcceptButton = btnOk;
- CancelButton = btnCancel;
- FormBorderStyle = FixedDialog;
- MaximizeBox = False;
- MinimizeBox = False;
- ShowInTaskBar = False;
- StartPosition = CenterParent;
- Text = "Фільтр".

Після розміщення на формі компонентів для них слід задати Tab-порядок, який визначає послідовність переходів між елементами керування при натисканні клавіші Tab. Для цього слід вибрати команду меню View\ Tab Order, і послідовно проклацати мишкою по всіх компонентах у дизайнері форм у

потрібному порядку. Якщо порядок задано, слід ще раз вибрати команду меню View\ Tab Order.

Для запам’ятовування числових значень площ додамо у клас FormFilter два поля – AreaMin та AreaMax (лістинг 22).

Лістинг 22

```
public partial class FormFilter : Form
{
    public double AreaMin;
    public double AreaMax;
```

Перепишемо конструктор так, щоб йому можна було передавати значення площ (лістинг 23).

Лістинг 23

```
public FormFilter(double areaMin, double areaMax)
{
    AreaMin = areaMin;
    AreaMax = areaMax;

    InitializeComponent();
}
```

При відображення форми FormFilter на екрані проведемо ініціалізацію її полів – заповнимо їх переданими конструктору значеннями. Ініціалізацію зручно проводити в оброблювачі події Load форми (лістинг 24).

Лістинг 24

```
private void fFilter_Load(object sender, EventArgs e)
{
    tbAreaMin.Text = AreaMin.ToString("0");
    tbAreaMax.Text = AreaMax.ToString("0");
}
```

При натисканні на кнопку "Ok" слід перевірити введені користувачем значення мінімальної та максимальної площі, перетворити їх в числовий тип даних і запам’ятати. Це здійснює оброблювач події Click кнопки (лістинг 25). Дії, які виконує цей метод, аналогічні до тих, що описані для кнопки "Ok" у вікні FormTown (лістинг 12).

Лістинг 25

```
private void btnOk_Click(object sender, EventArgs e)
{
    if (!double.TryParse(tbAreaMin.Text, out AreaMin))
    {
        MessageBox.Show("Неправильно введено число!", "",
            MessageBoxButtons.OKCancel, MessageBoxIcon.Warning);
        tbAreaMin.Focus();
        return;
    }
    if (!double.TryParse(tbAreaMax.Text, out AreaMax))
    {
        MessageBox.Show("Неправильно введено число!", "",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        tbAreaMin.Focus();
        return;
    }

    DialogResult = DialogResult.OK;
}
```

Код у файлі FormFilter.cs використовує такі простори імен:

```
using System;
using System.Windows.Forms;
```

### 2.4.11.2. Програмування кнопки "Фільтрування даних"

Відкриємо головну форму в дизайнері. В оброблювачі події Click кнопки "Фільтрування даних" реалізуємо такий алгоритм:

- перевіряємо, чи є у джерелі даних якісь дані. Якщо даних немає, припиняємо роботу методу;
- проходимо по всіх елементах Town джерела даних і знаходимо мінімальне та максимальне значення площі міста у переліку;
- створюємо вікно фільтра FormFilter, передавши йому знайдені мінімальне та максимальне значення площ;
- показуємо вікно фільтра на екрані;
- якщо користувач підтвердив операцію фільтрування, то пройти по всіх елементах списку джерела даних, перевірити для кожного об’єкта Town значення площі, і якщо площа виходить за вказаний користувачем діапазон, видалити відповідний запис про місто з джерела даних;
- при видаленні даних вміст таблиці у головному вікні автоматично оновиться, і воно не міститиме записів, які не відповідають заданим критерія площі.

Цей алгоритм запрограмовано у лістингу 26.

Лістинг 26

```
private void btnFilter_Click(object sender, EventArgs e)
{
    if (bindSrcTowns.List.Count == 0) return;

    double areaMin = ((Town)bindSrcTowns.List[0]).Area;
    double areaMax = ((Town)bindSrcTowns.List[0]).Area;

    foreach(Town town in bindSrcTowns.List)
    {
        if (town.Area < areaMin) areaMin = town.Area;
        if (town.Area > areaMax) areaMax = town.Area;
    }

    FormFilter ft = new FormFilter(areaMin, areaMax);
    if (ft.ShowDialog() == DialogResult.OK)
    {
        StartOfLoop:
        for (int i = 0; i < bindSrcTowns.Count; i++)
        {
            if((((Town)bindSrcTowns.List[i]).Area < ft.AreaMin) ||
               (((Town)bindSrcTowns.List[i]).Area > ft.AreaMax))
            {
                bindSrcTowns.RemoveAt(i);
                goto StartOfLoop;
            }
        }
    }
}
```

### 2.4.11.3. Проміжне тестування

Запустіть застосунок і завантажте раніше збережені дані. Подивіться на значення площ для наявних у таблиці міст. Якщо площі для всіх записів одинакові, то відредагуйте їх так, щоб вони відрізнялися. Тоді натисніть на кнопку "Фільтрування даних". Має з’явитися вікно для введення параметрів фільтрування, у якому початкові мінімальне та максимальне значення фільтра буде задано найменше та найбільше значення площ міст, наявних у списку в даний момент. Перевірте, чи це так.

Введіть інші значення мінімальної та максимальної площі для фільтра, вибравши їх так, щоб частина записів мала площу, яка не вкладається в цей діапазон. Підтвердіть операцію фільтрування, натиснувши кнопку "Ok" у вікні FormFilter. Перевірте, чи правильно відфільтровано дані.

### 2.4.12. Сортування даних

Сортування записів про міста буде виконуватися при виборі користувачем елемента списку tscbSortBy, розташованого на панелі інструментів головного вікна. Елементи цього списку (сортувати за: "назвою", "країною" тощо) задають параметр, за яким слід сортувати дані у таблиці. Саме сортування здійснюється в оброблювачі події SelectedIndexChanged компонента tscbSortBy.

Алгоритм сортування такий:

- перевіряємо, чи є у джерелі даних якісь дані. Якщо даних немає, припиняємо роботу методу;
- аналізуємо, який елемент списку tscbSortBy вибрано, і залежно від цього встановлюємо значення змінної SortOrder (цю властивість ми додали до класу форми самостійно). Тип змінної SortOrder – перелік, який приймає значення SortByName, SortByCountry тощо;
- проходимо у циклі по всіх елементах Town у списку джерела даних і задаємо для кожного елемента порядок сортування SortOrder. Це потрібно для методу CompareTo() класу Town;
- копіюємо список міст з джерела даних у новий список towns;
- сортуємо список towns, викликавши метод Sort(). Цей метод використовує метод CompareTo() класу Town, який спадкується від інтерфейсу IComparable.
- копіюємо список towns у джерело даних. Дані автоматично відображуються у таблиці;
- активуємо компонент таблиці gvTowns, викликавши його метод Focus().

Цей алгоритм реалізує лістинг 27.

Лістинг 27

```
private void tscbSortBy_SelectedIndexChanged(object sender,
    EventArgs e)
{
    if (bindSrcTowns.List.Count == 0) return;

    switch (tscbSortBy.SelectedIndex)
    {
        case 0:
            SortOrder = TownsSortOrder.SortByName;
            break;
        case 1:
            SortOrder = TownsSortOrder.SortByCountry;
            break;
        case 2:
            SortOrder = TownsSortOrder.SortByRegion;
            break;
```

```
            case 3:
                SortOrder = TownsSortOrder.SortByPopulation;
                break;
            case 4:
                SortOrder = TownsSortOrder.SortByArea;
                break;
            case 5:
                SortOrder = TownsSortOrder.SortByLatitude;
                break;
            case 6:
                SortOrder = TownsSortOrder.SortByLongitude;
                break;
            default:
                SortOrder = TownsSortOrder.SortByName;
                break;
        }
        foreach(Town town in bindSrcTowns.List)
            town.SortOrder = SortOrder;

        List<Town> towns = new List<Town>();

        foreach (Town town in bindSrcTowns.List)
            towns.Add(town);

        towns.Sort();

        bindSrcTowns.Clear();
        foreach (Town town in towns)
            bindSrcTowns.Add(town);

        gvTowns.Focus();
    }
```

Після того, як алгоритм сортування реалізовано, слід розкоментувати останній рядок коду із лістингу 19, в якому викликається щойно розроблений метод tscbSortBy_SelectedIndexChanged(). Для цього слід видалити символи коментаря ("//") на початку рядка. Це забезпечить сортування даних відразу після їх відкриття із файлу.

### 2.4.12.1. Проміжне тестування

Запустіть застосунок і завантажте раніше збережені дані. Спробуйте відсортувати дані, вибравши зі списку tscbSortBy різні параметри сортування. Якщо даних недостатньо, введіть ще кілька записів, щоб можна було перевірити правильність сортування за кожним параметром. Перевірте, чи дані сортуються правильно, для всіх параметрів сортування.

### 2.4.13. Пошук

Пошук міста з потрібною назвою має відбуватися при введенні рядка з назвою в поле "Знайти:" tstbSearch. Пошук проводитиметься при введені у поле пошуку кожного символу. Операцію пошуку запрограмуємо в оброблювачі події TextChanged компонента tstbSearch. Для цього реалізуємо такий алгоритм (лістинг 28):

- перевіряємо, чи є у джерелі даних якісь дані. Якщо даних немає, припиняємо роботу методу;
- проходимо у циклі по всіх записах джерела даних, і перевіряємо, чи назва міста починається з введеного в поле пошуку значення. Якщо так, то поточною активною коміркою таблиці gvTowns встановлюємо першу комірку відповідного рядка.

Лістинг 28

```
private void tstbSearch_TextChanged(object sender, EventArgs e)
{
    if (bindSrcTowns.List.Count == 0) return;

    string findText = tstbSearch.Text;
    if (findText.Trim() == "") return;
    gvTowns.ClearSelection();
    CultureInfo culture = new CultureInfo("uk-UA");

    for (int i = 0; i < bindSrcTowns.Count; i++)
    {
        if (((Town)bindSrcTowns.List[i]).Name.StartsWith(
            findText, true, culture))
        {
            gvTowns.CurrentCell = gvTowns.Rows[i].Cells[0];
            break;
        }
    }
}
```

Метод StartsWith класу String, використаний в лістингу 28, повертає true, якщо рядок починається із вказаного значення. Щоб задати активну (поточну) комірку таблиці (компонента gvTowns), встановлюємо її властивість CurrentCell.

#### 2.4.13.1. Проміжне тестування

Запустіть застосунок, відкрийте попередньо збережені дані, і почніть вводити у поле пошуку назву одного з існуючих у списку міст. Рядок з цим містом має стати активним. Перевірте роботу механізму пошуку для різних записів.

### 2.4.14. Припинення роботи застосунку

Припинення роботи застосунку завжди супроводжується закриванням його головного вікна. При закриванні форми виникає подія FormClosing. Зокрема, вона виникає при натисканні на кнопку з хрестиком у заголовку вікна. Використаємо цю подію для перепитування необхідності закрити застосунок (лістинг 29).

Лістинг 29

```
private void FormMain_FormClosing(object sender,
    FormClosingEventArgs e)
{
    if (MessageBox.Show("Закрити застосунок?", "Вихід з програми",
        MessageBoxButtons.OKCancel,
        MessageBoxIcon.Question) == DialogResult.OK)

        e.Cancel = false;
    else e.Cancel = true;
}
```

В оброблювач цієї події передається об'єкт e класу FormClosingEventArgs, який має властивість Cancel (скасувати) логічного типу. Якщо цій властивості присвоїти значення true, то операція закривання вікна буде скасована, і воно продовжить свою роботу. У методі з лістингу 29, якщо користувач не підтвердив своє бажання припинити роботу із застосунком, цій властивості присвоюється значення true.

Щоб можна було припинити роботу застосунку при натисканні кнопки btnExit ("Вийти з програми") на панелі інструментів головного вікна, в оброблювачі події Click для цієї кнопки запишемо простий код з лістингу 30. Метод Exit() класу Application надсилає головному вікну застосунку системне повідомлення про необхідність припинити роботу. При цьому виникає подія FormClosing і спрацьовує код з лістингу 29: застосунок перепитує необхідність припинити роботу. Якщо користувач скасує операцію виходу, діяльність програми триватиме.

Лістинг 30

```
private void btnExit_Click(object sender, EventArgs e)
{
     Application.Exit();
}
```

#### 2.4.14.1. Проміжне тестування

Запустіть застосунок і спробуйте його закрити різними способами (через кнопку в заголовку вікна і через кнопку на панелі інструментів). Перевірте, чи правильно веде себе програма.

### 2.4.15. Відображення додаткової інформації у рядку стану

При проектуванні рядка стану ми передбачили у ньому три поля для додаткових даних:

- для назви відкритого файлу (це поле заповнюється при читанні даних з файлу, див. відповідні лістинги);
- для густини населення вибраного міста;
- для річного доходу, який припадає на одного мешканця міста.

Щоб відобразити додаткову інформацію про місто з виділеного у таблиці рядка, використаємо подію CurrentCellChanged таблиці gvTowns (лістинг 31). В оброблювачі цієї події спочатку отримуємо об'єкт town (класу Town), який відповідає поточному рядку таблиці. Далі викликаємо його методи PopulationDensity() та GetYearIncomePerInhabitant(), які розраховують відповідно густину населення і дохід на мешканця. Ці методи описані в розділі, присвяченому розробці класу Town. Обидва ці методи повертають дійсне число, яке форматуємо і виводимо у відповідні компоненти рядка стану.

Лістинг 31

```
private void gvTowns_CurrentCellChanged(object sender,
    EventArgs e)
{
    Town town = (Town)bindSrcTowns.List[bindSrcTowns.Position];
    tsslPopDensity.Text =
        string.Format("Густина населення: {0:0.000} чол/кв.км",
        town.PopulationDensity());
    tsslYearIncomePerInhabitant.Text =
        string.Format("Річний дохід на мешканця: {0:0} грн",
        town.GetYearIncomePerInhabitant());
}
```

#### 2.4.15.1. Проміжне тестування

Запустіть застосунок, і, активуючи різні записи у таблиці, прослідкуйте, як змінюються дані в рядку стану. Перевірте, чи правильно відображуються додаткові дані.

### 2.4.16. Збереження налаштувань застосунку

При закінченні сесії роботи застосунку зберігатимемо такі налаштування:

- ширину головного вікна;
- висоту головного вікн;
- відступ головного вікна від лівого краю екрану;
- відступ головного вікна від верхнього краю екрану;
- порядок сортування записів у таблиці.

C# пропонує для вирішення подібних завдань клас Settings. Спочатку слід створити параметри для налаштувань. Для цього у вікні Solution Explorer потрібно розгорнути елемент Properties і двічі клацнути на файлі Settings.settings. Саме у цьому файлі за замовчуванням зберігається опис параметрів, які слід зберігати. При проектуванні застосунку можна створити й інші файли .settings для створення різних груп налаштувань. Але ми скористаємось вже наявним файлом.

Після подвійного клацання на файлі Settings.settings у Visual Studio відкриється редактор налаштувань (рис. 13). Кожен параметр має:

- назву (стовпчик Name);
- тип (стовпчик Type);
- область дії (стовпчик Scope);
- значення (стовпчик Value).

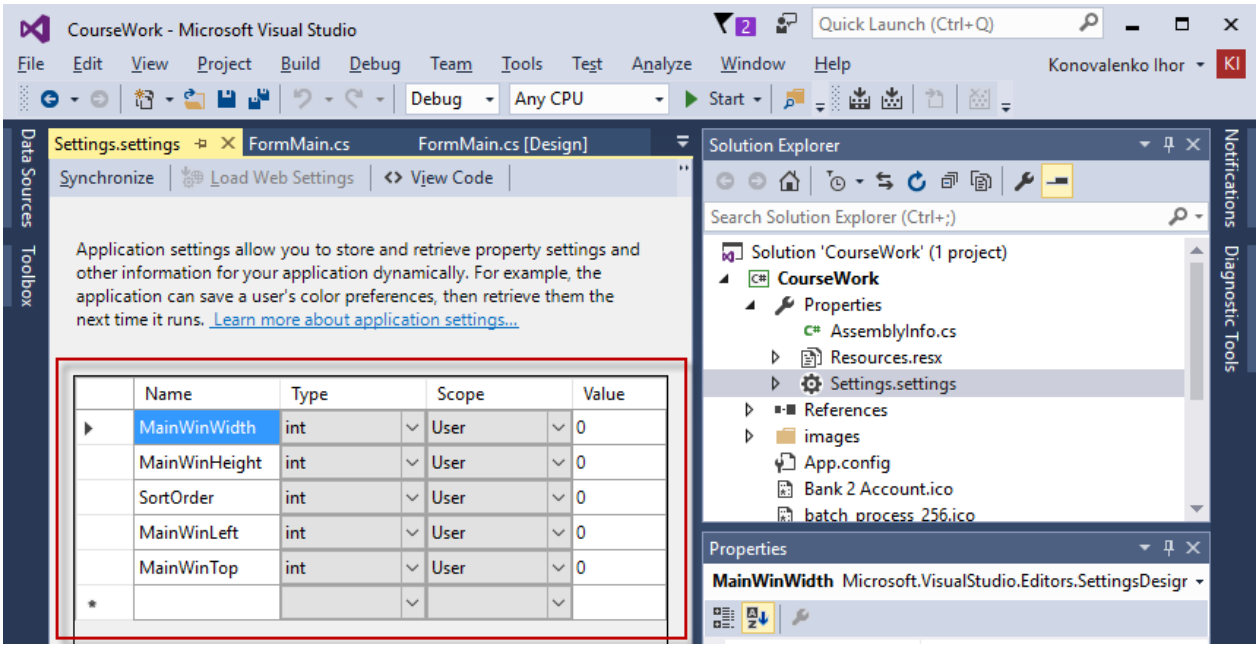


Рис. 13. Створення параметрів для налаштувань застосунку (параметри виділено червоною рамкою)

Кожен параметр представлений окремим рядком таблиці (рис. 13). Створимо параметри відповідно до таблиці 4. Тип всіх параметрів – ціле число (int), а область дії – User. Ця область дії означає, що налаштування можуть бути записані та прочитані програмно. Можлива ще інша область дії – Application. Налаштування цього виду створюються при розробці застосунку і програмно можуть бути тільки прочитані. Записати у них якесь значення програмно неможливо.

Таблиця №4. Параметри для збереження

| Назва (Name) | Тип (Type) | Область дії (Scope) |
|---|---|---|
| MainWinWidth | int | User |
| MainWinHeight | int | User |
| SortOrder | int | User |
| MainWinLeft | int | User |
| MainWinTop | int | User |

Всі створені таким чином налаштування супроводжуються додаванням до властивості Default класу Settings поля з заданою назвою та типом. Таким чином, доступ до параметру MyParameter у налаштуваннях здійснюється так: Settings.Default.MyParameter.

Зберігати налаштування потрібно при виході з програми. Для цього зручно використовувати подію FormClosed головного вікна. Отже, створимо оброблювач цієї події і приведемо його до виду згідно лістингу 32.

Лістинг 32

```
private void FormMain_FormClosed(object sender,
    FormClosedEventArgs e)
{
    Properties.Settings.Default.MainWinWidth = Width;
    Properties.Settings.Default.MainWinHeight = Height;
    Properties.Settings.Default.MainWinLeft = Left;
    Properties.Settings.Default.MainWinTop = Top;
    Properties.Settings.Default.SortOrder =
        tscbSortBy.SelectedIndex;
    Properties.Settings.Default.Save();
}
```

Читати налаштування зручно при створенні головного вікна. Тому в оброблювач події Load головної форми (він у нас вже створений і містить код з

лістингу 7) слід на початку дописати код, який читатиме налаштування і присвоюватиме прочитані значення відповідним елементам головного вікна (лістинг 33).

Лістинг 33

```
if (Properties.Settings.Default.MainWinWidth > 0)
    Width = Properties.Settings.Default.MainWinWidth;
if (Properties.Settings.Default.MainWinHeight > 0)
    Height = Properties.Settings.Default.MainWinHeight;
if (Properties.Settings.Default.MainWinLeft > 0)
    Left = Properties.Settings.Default.MainWinLeft;
if (Properties.Settings.Default.MainWinTop > 0)
    Top = Properties.Settings.Default.MainWinTop;

tscbSortBy.SelectedIndex = Properties.Settings.Default.SortOrder;
```

Налаштування, які зберігають описаним у цьому пункті способом, записуються у форматі xml у файл user.config, який розташований за шляхом:

[X]:\Users\[UserName]\AppData\Local\[ProjectName]\[ProjectName].exe_[Hash]\[Version]\

Якщо відкрити його у текстовому редакторі (наприклад, у "Блокноті"), то можна побачити поточні значення параметрів.

### 2.4.16.1. Проміжне тестування

Запустіть застосунок, змініть розміри та положення головного вікна. Закрийте його, запустіть ще раз, і перевірте, чи розміри і положення відповідають тим, які були в кінці минулої сесії.

---

На цьому етапі розробки всі операції, які має виконувати застосунок згідно завдання, вже реалізовано. Залишилося ще "під’єднати" вже спроектований код до відповідних елементів меню.

## 2.4.17. Призначення методів для елементів головного меню

Команди меню дублюють операції, які виконують кнопки на панелі інструментів. Так, команда меню Файл\Відкрити… повинна виконувати ті ж дії, що і кнопка "Завантажити з бінарного файлу" панелі.

Повторно писати (чи навіть копіювати) той самий код для меню не потрібно. Кожному елементу меню як оброблювач події Click слід призначити вже розроблений оброблювач такої ж події відповідної кнопки.

Перейдемо в дизайнер головної форми, і, клацнувши на головному меню, відкриємо редактор меню. Далі слід виділити елемент меню miOpen (команда "Відкрити…"), перейти до події Click, натиснути кнопку зі стрілкою, і з переліку методів вибрати метод btnOpenFromBinary_Click.

Тоді, під час роботи застосунку, при виборі цієї команди меню спрацює той же код, що і при натисканні кнопки btnOpenFromBinary.

Для елемента меню miSave (команда "Зберегти…") слід задати метод btnSaveAsBinary_Click і т.д. Подібні маніпуляції слід зробити над всіма елементами меню, згідно таблиці 5.

Таблиця №5. Оброблювачі події Click для елементів меню

| Елемент меню | Назва елемента меню | Метод для події Click |
|---|---|---|
| Відкрити… | miOpen | btnOpenFromBinary_Click |
| Зберегти… | miSave | btnSaveAsBinary_Click |
| Імпортувати текст… | miImport | btnOpenFromText_Click |
| Експортувати текст… | miExport | btnSaveAsText_Click |
| Вихід | miExit | btnExit_Click |
| Додати… | miAdd | btnAdd_Click |
| Змінити… | miEdit | btnEdit_Click |
| Видалити | miDelete | btnDel_Click |
| Очистити всі | miClear | btnClear_Click |
| Фільтрувати… | miFilter | btnFilter_Click |

Коли методи для всіх елементів головного меню призначені, слід призначити методи для контекстного меню. Це робиться повністю аналогічно, як і для головного меню.

### 2.4.17.1. Програмування команди меню "Про програму…"

Це єдиний елемент меню, який не має відповідника серед кнопок панелі інструментів. Для нього слід створити оброблювач події Click і запрограмувати код згідно лістингу 34. Цей код просто показує повідомлення про автора програми. На місці фрази [ПРІЗВИЩЕ І.П.] слід вказати своє прізвище та ініціали.

Лістинг 34

```
private void miAbout_Click(object sender, EventArgs e)
{
    MessageBox.Show("Застосунок демонструє зразок " +
        "виконання курсової роботи\n" +
        "з курсу 'Об'єктно-орієнтоване програмування'\n\n" +
        "Розробив: доц. каф. АВ Тернопільського національного " +
        "технічного університету \n[ПРІЗВИЩЕ І.П.]",
        "Про програму", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}
```

### 2.4.17.2. Проміжне тестування

Запустіть застосунок і протестуйте роботу всіх команд головного та контекстного меню.

### 2.4.18. Кінцеве тестування

Тестування є важливим етапом розробки програмного забезпечення.

Запустіть розроблений застосунок. Введіть не менше 10 записів з реальними (або правдоподібними) даними. Повторно протестуйте всі функції застосунку. Якщо якась із них працює не так як слід, поверніться до відповідного розділу і перевірте, чи все зроблено вірно: чи задано компонентам форми всі потрібні властивості, чи створено відповідні методи (оброблювачі подій), і чи правильно введено програмний код. На рис. 14 показано вигляд головного вікна застосунку з введеними даними.

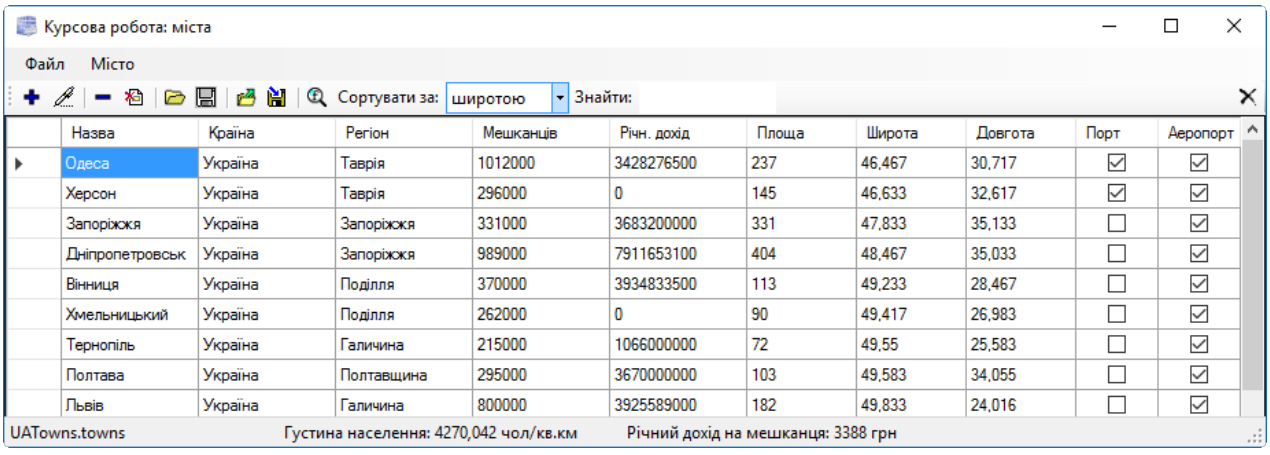


Рис. 14. Головне вікно розробленого застосунку з даними

Повністю готовий проект застосунку, який описаний у даних методичних вказівках, можна завантажити з матеріалів електронного навчального курсу "Об’єктно-орієнтоване програмування". Він розташований в архіві _CourseWork.zip. Його можна розпакувати, відкрити в середовищі Microsoft Visual Studio та детально розглянути роботу застосунку.