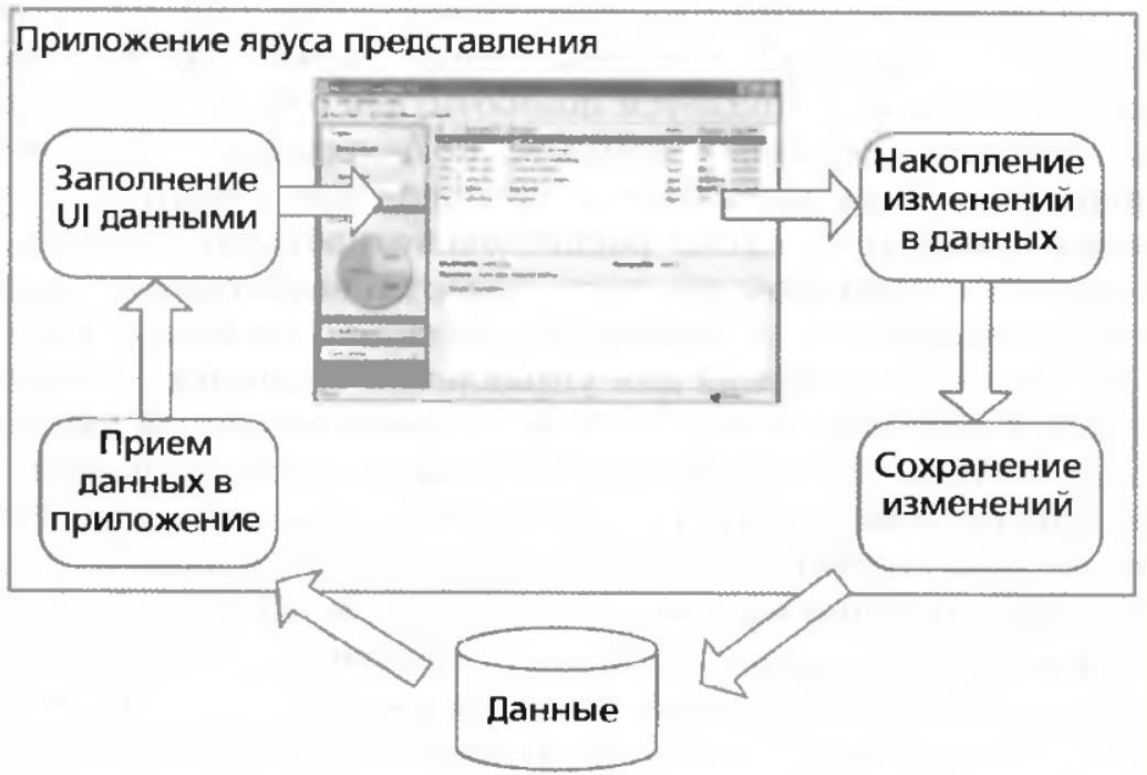
ТЕХНОЛОГІЯ ПРИВ’ЯЗУВАННЯ ДАНИХ

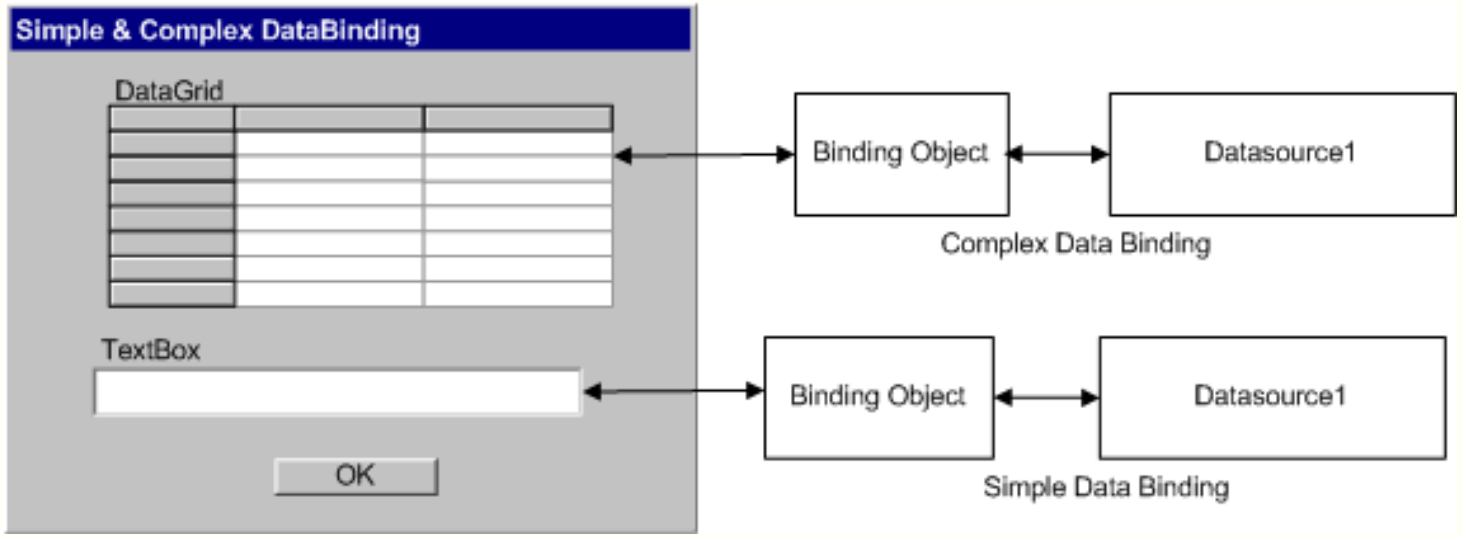
Питання 14.3.

Проблема прив’язування даних



- У минулому, якщо вам потрібно показати дані користувачеві, ви повинні були отримати ці дані зі сховища і прийняти їх у проектованому додатку.
  - Необхідно було написати власний програмний код для подання цих даних в графічному вигляді або відобразити ці дані за допомогою стандартних елементів управління, налаштовуючи їх відповідно до відображеними даними.
  - Необхідно було також подбати про код, записуючого змінені дані в сховище, з якого взяті вихідні дані для відображення.
- ***Прив'язка до даних (data binding)*** інкапсулює всі ці операції в компонентах, які допомагають відображати дані, що дозволяє зменшити обсяг підлягає написанню коду.
  - Частина цього коду йде в візуальні елементи управління, а частина - в приховані компоненти, спрощуючи прив'язку до даних.

Прив’язка даних у Windows Forms



- Прив’язка даних надає змогу візуальним елементам клієнтського коду (TextBox, Datagrid та ін.) під’єднуватись до джерел даних (datasource), таких як DataSet-и, DataView-и, масиви тощо.
  - Двостороннє підключення (connection) встановлюється так, щоб будь-які зміни в джерелі даних негайно відображались на візуальному елементі, і навпаки.

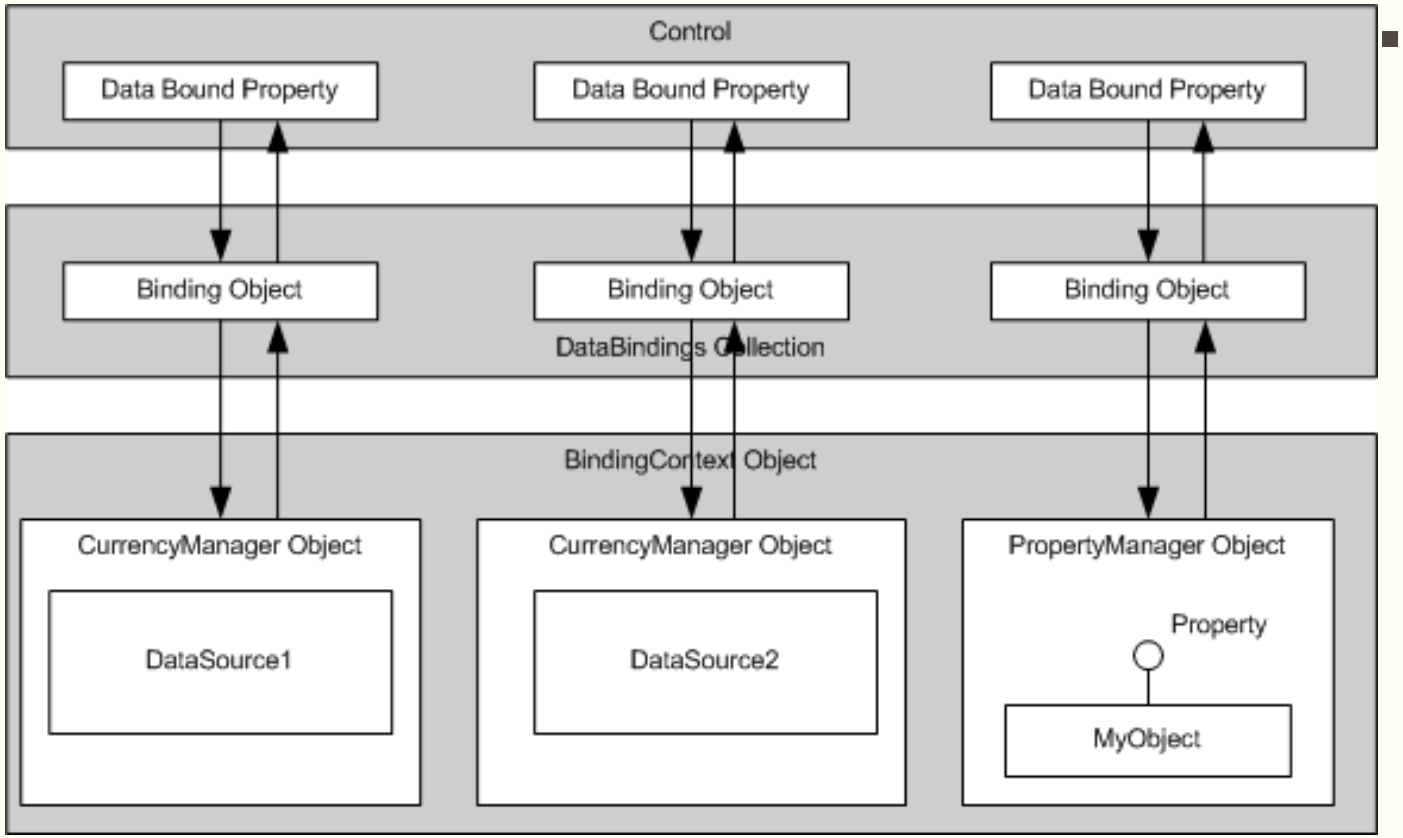
Переваги та недоліки прив’язування даних

- Переваги:
  - Прив’язування даних може використовуватись, щоб швидко писати керовані даними додатки. Прив’язка даних .NET дозволяє писати менше коду та швидше його виконувати, проте still get the work done in the best way.
  - .NET автоматично утворює багато коду, який стосується прив’язування даних, тому розробникам не потрібно витрачати час на базову реалізацію прив’язки. but still has the flexibility of modifying any code that he would like to. We get the benefits of bound as well as unbound approach.
  - Процес прив’язування даних може управлятись подіями.
- Недоліки
  - Можна писати більш оптимізований код, використовуючи unbound або традиційні методи.
  - Загальна гнучкість може реалізовуватись лише за допомогою unbound approach.

Поняття в контексті прив’язування даних

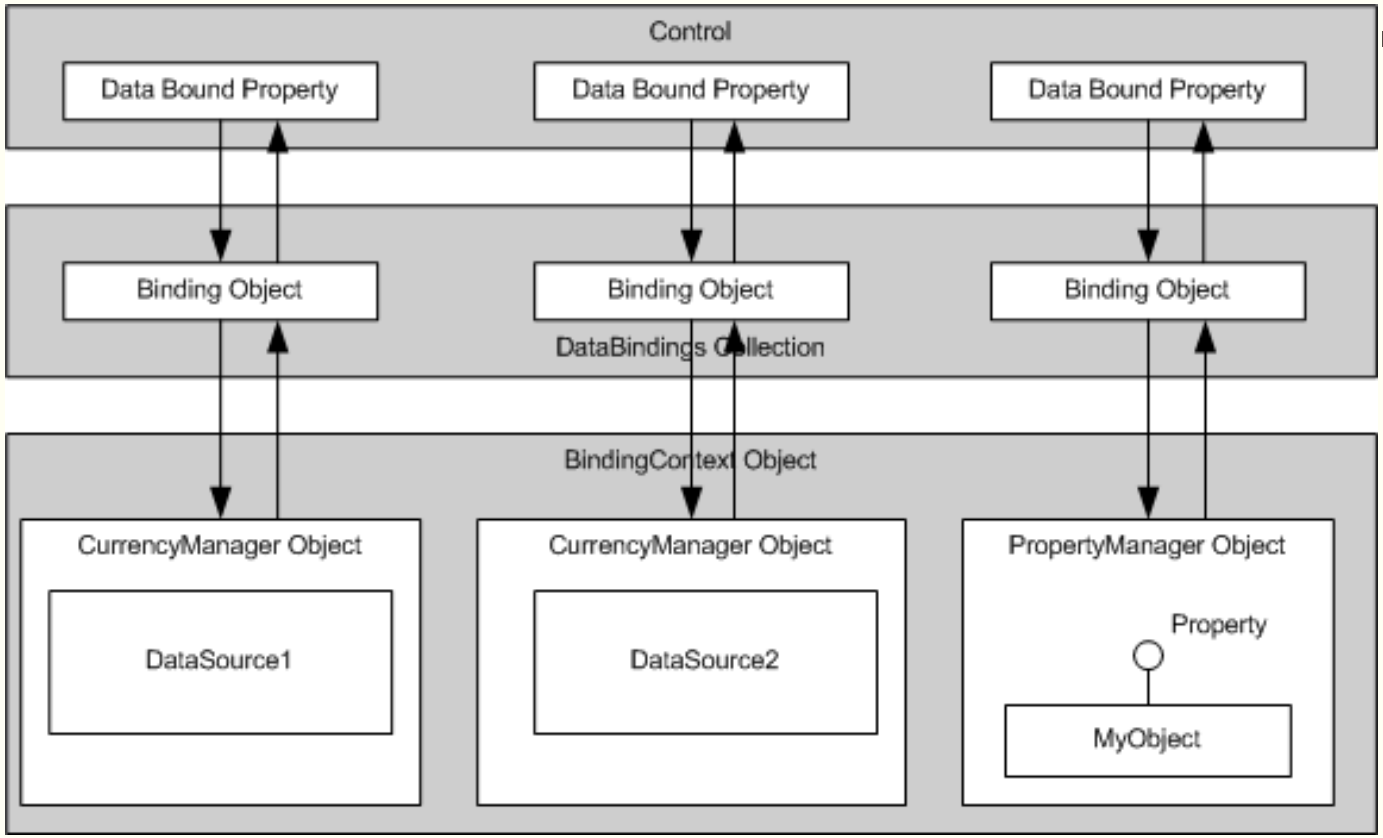
- Щоб прив’язування даних відбулось, постачальник даних (data provider) та споживач даних (data consumer) мають існувати з синхронним зв’язком між ними.
  - Постачальники даних мають дані, а споживачі використовують дані, видані постачальниками, та відображають їх.
- .NET розширяє перелік можливих постачальників даних: будь-який клас чи компонент, який реалізує інтерфейс IList є валідним DataSource-ом:
  - Масиви (Arrays)
  - DataColumn
  - DataTable
  - DataView
  - DataSet
- Зауважте, що інтерфейс IList дозволяє прив’язуватись тільки в рантаймі.
  - Якщо потрібно підтримувати прив’язування на етапі дизайну, необхідно реалізовувати ще й інтерфейс IComponent. Також зауважте, що неможливо прив’язуватись до DataReader-ів у Windows Forms (можна у web forms).
- .NET Framework підтримує просте та складне прив’язування даних.
  - Проста прив’язка підтримується елементами управління на зразок TextBox: лише одне значення даних може відображатись елементом управління за раз.
  - Складна (complex) прив’язка підтримується елементами управління на зразок DataGrid і допускає відображення понад одного значення з DataSource.

Переміщення даних протягом DataBinding



- Елементи управління в .NET можуть мати багато властивостей, які можливо прив'язати до DataSource.
  - Кожна прив'язана властивість пов'язана з Binding-об'єктом.
  - Оскільки елемент управління може мати багато Binding-об'єктів, працює колекція (екземпляр ControlBindingsCollection) з усіма Binding-об'єктами.
  - Пам'ятайте, що різні властивості одного елементу управління можуть прив'язуватись (bound) до різних джерел даних.

Переміщення даних протягом DataBinding



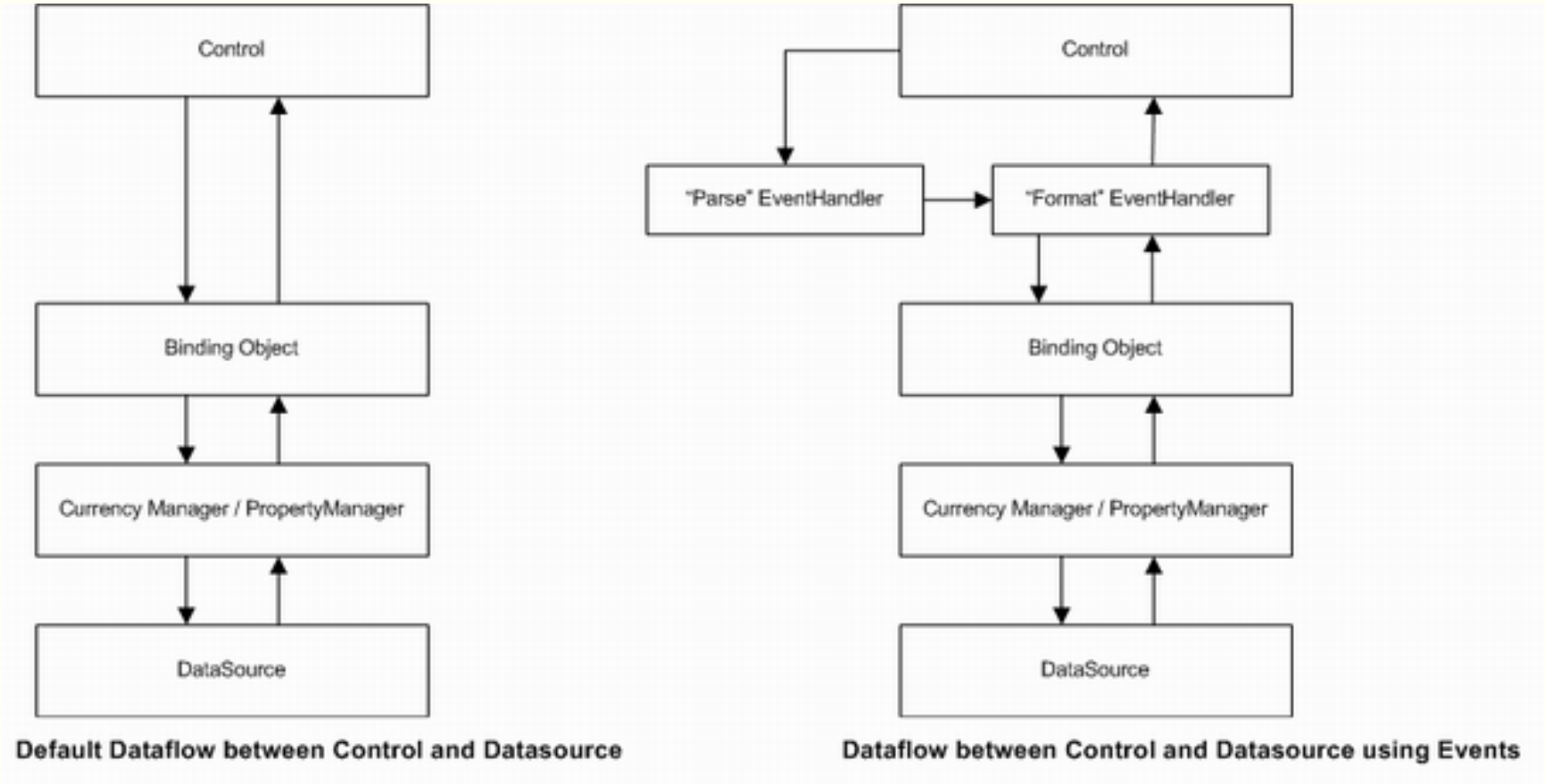
- Кожний Binding-об'єкт спілкується з CurrencyManager або PropertyManager.
  - Класи CurrencyManager та PropertyManager породжені від базового класу BindingManagerBase, мета якого – підтримувати the concurrency між джерелом даних та елементом управління.
  - Клас CurrencyManager використовується, коли джерело даних реалізує інтерфейс IList.
  - Приклади подібних джерел: DataView, DataSet, ArrayList тощо.
  - CurrencyManager може використовуватись як для простого, так і складного прив'язування даних.
  - Проте клас PropertyManager застосовується, коли джерело даних є екземпляром user-defined класу.

Зміна зовнішнього вигляду та поведінки форми

- ***Рекомендація:*** якщо клас має бути джерелом даних, краще використовувати CurrencyManager, якщо клас є контейнером даних.
  - Проте при потребі прив'язування елементу управління до властивостей, розкритих вашим власним класом, простіше застосувати PropertyManager, оскільки не потрібно реалізовувати інтерфейс IList.
- Оскільки форма може містити багато елементів управління кожний, з яких прив'язаний до іншого джерела, потрібний клас для управління об'єктами CurrencyManager та PropertyManager.
  - Тому кожна форма Windows у .NET має за умовчанням BindingContext-об'єкт, пов'язаний з нею.
  - Також ви можете завжди створити більше BindingContext-об'єктів на формі.
  - Об'єкт BindingContext – це колекція об'єктів CurrencyManager та PropertyManager.
- У підсумку:
  - Елемент управління (control) може мати багато доступних для прив'язування властивостей.
  - Кожна прив'язана властивість елементу управління пов'язана з Binding-об'єктом.
  - Усі Binding-об'єкти для елементу управління розміщуються всередині властивості DataBindings елементу управління, екземпляр якого є об'єктом класу ControlBindingsCollection.
  - Each databinding object talks to a CurrencyManager or PropertyManager object.
  - CurrencyManager and PropertyManager are derived from the BindingManagerBase class.
  - The BindingContext object is a collection of CurrencyManager and PropertyManager objects.
  - By default, a form contains one BindingContext object. More BindingManagerBase objects can be created and added to the BindingContext collection.
  - Each CurrencyManager or PropertyManager encapsulates the data access to one datasource per BindingContext object.

Controlling DataBinding

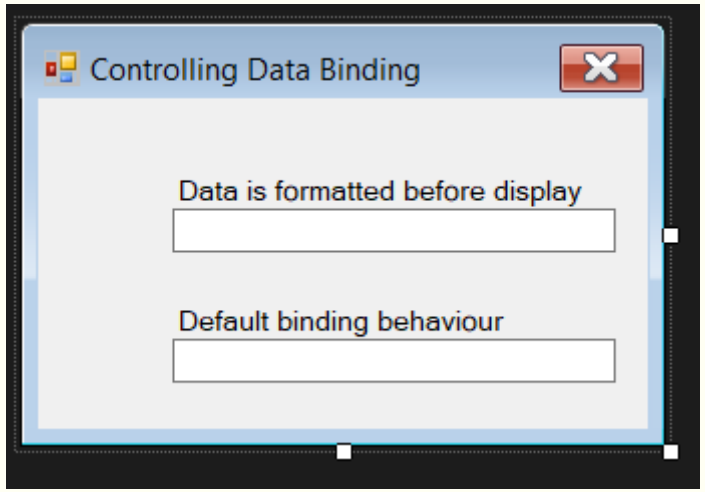
- The real flexibility and power of databinding in .NET is realized because the Binding and BindingManagerBase classes supports events.
  - This enables us to change the data passed between the Control and the datasource.

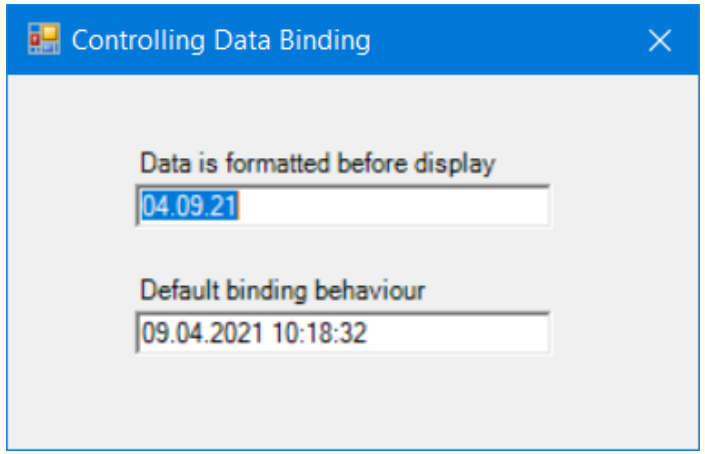


Компонування інтерфейсу за допомогою контейнерних елементів управління

- Binding-об’єкт розкриває 2 події: Format і Parse.
  - Подія Format спрацьовує двічі: спочатку при передачі даних від джерела до елемента управління, а потім при зміні даних джерела та оновленні елементу управління.
  - Подія Parse спрацьовує, коли дані витягуються з елементу управління в джерело даних.
- The Currency Manager (derived from the BindingManagerBase class) exposes three events: CurrentChanged, PositionChanged and ItemChanged. CurrentChanged is triggered when the bound value changes; PositionChanged is triggered when the position property has changed and ItemChanged is triggered when the current item has changed. Please note that the PropertyManager class supports only 2 events: CurrentChanged, PositionChanged.
- These events enable a user to have fine control over the dataflow from the control to the datasource and vice versa.

Приклад простого прив'язування даних

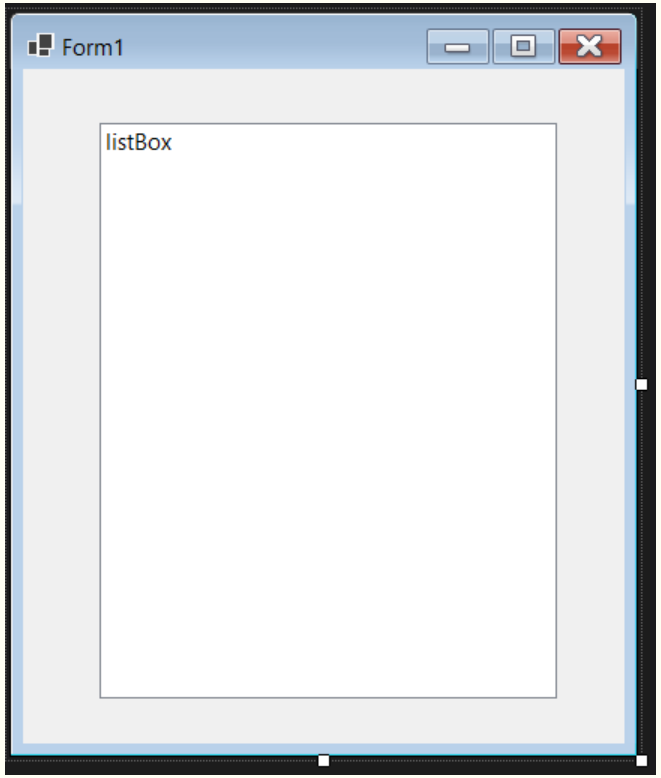




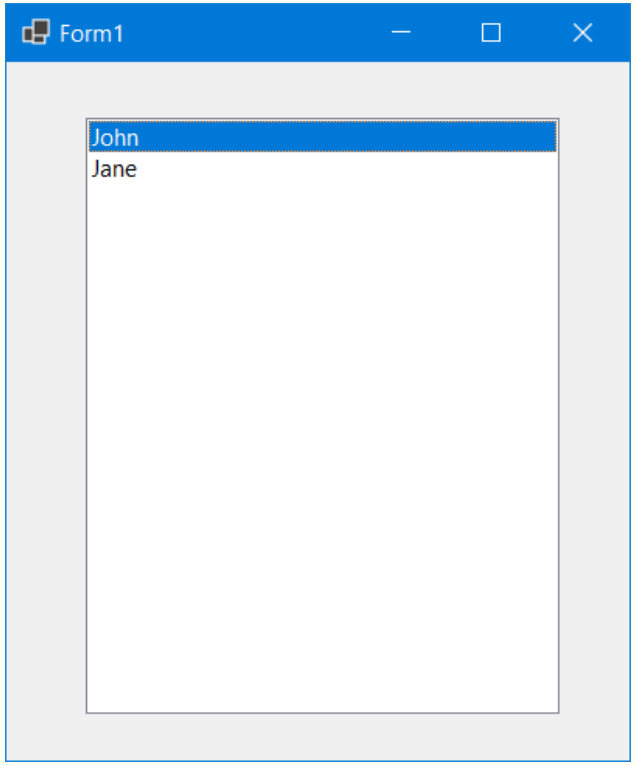

```
private MyDateTime oDt = new MyDateTime();
private Binding oBinding;

private void Form1_Load(object sender, System.EventArgs e)
{
    oBinding = new Binding("Text", oDt, "GetDateTime");
    //The event handler is added
    oBinding.Format += new
                    System.Windows.Forms.ConvertEventHandler(this.oBinding_Format);

    txt1.DataBindings.Add(oBinding);
    txt2.DataBindings.Add("Text", oDt, "GetDateTime");
}

private void oBinding_Format(object sender, System.Windows.Forms.ConvertEventArgs e)
{
    e.Value = ((DateTime)e.Value).ToString("MM/dd/yy");
}
```

```
public class MyDateTime
{
    private DateTime dt;
    public MyDateTime() {
        this.dt = DateTime.Now;
    }

    public DateTime GetDateTime {
        get { return dt; }
        set { dt = value; }
    }
}
```

Прив'язування масиву до елементу ListBox

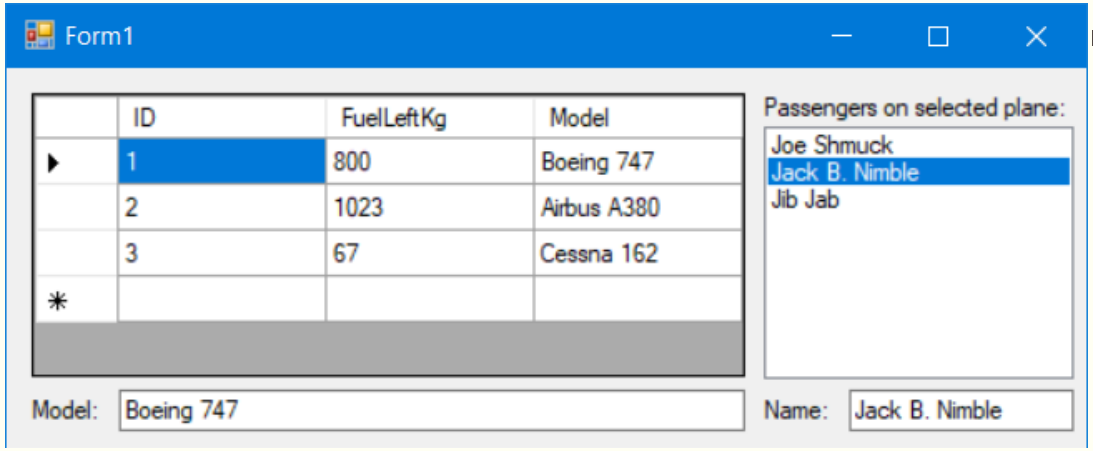



```
private void Form1_Load(object sender, EventArgs e)
{
     Person[] People = new Person[] {
          new Person("John","Smith"),
          new Person("Jane" ,"Doe")
     };
     listBox.DataSource = People;
     listBox.DisplayMember = "FirstName";
}


internal struct Person {
    private string firstName, lastName;

    public Person(string firstName, string lastName)
    {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public string FirstName {
        get { return firstName; }
    }

    public string LastName {
        get { return lastName; }
    }
}
```



- Якщо не задати DisplayMember, буде використане значення від методу ToString() (за умовчанням – "ListBoxBinding.Person", проте можна замістити реалізацію ToString()).

Синхронізація даних між візуальними елементами

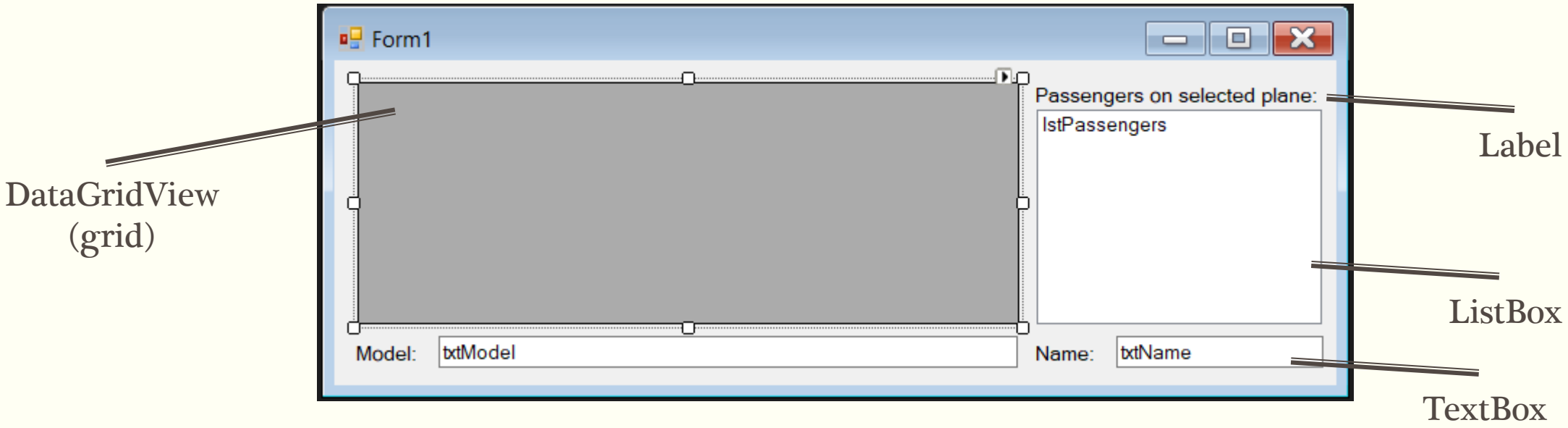


- Розглянемо приклад для літака з пасажирами.

```
class Airplane
{
    public Airplane() { _id = ++lastID; }
    public Airplane(string model, int fuelKg) {
       _id = ++lastID;
       Model = model;
       _fuelKg = fuelKg;
    }
    private static int lastID = 0;
    public int _id;
    public int ID { get { return _id; } }
    public int _fuelKg;
    public int FuelLeftKg { get { return _fuelKg; } set { _fuelKg = value; } }
    public string _model;
    public string Model { get { return _model; } set { _model = value; } }
    public List<Passenger> _passengers = new List<Passenger>();
    public List<Passenger> Passengers { get { return _passengers; } }
}
```

```
class Passenger
{
    public Passenger(string name) {
        _id = ++lastID;
        Name = name;
    }
    private static int lastID = 0;
    public int _id;
    public int ID { get { return _id; } }
    public string _name;
    public string Name { get { return _name; } set { _name = value; } }
}
```

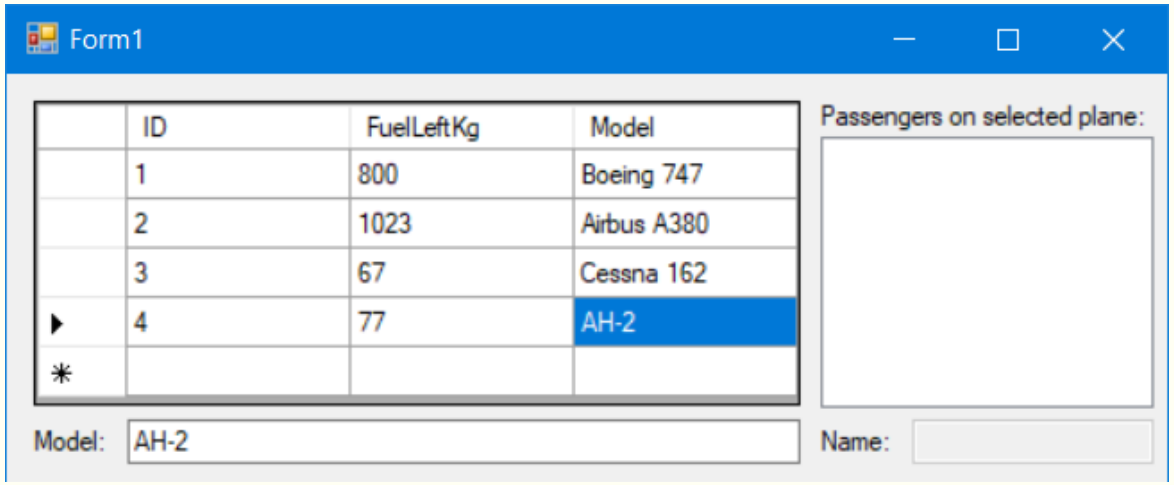
Синхронізація даних між візуальними елементами

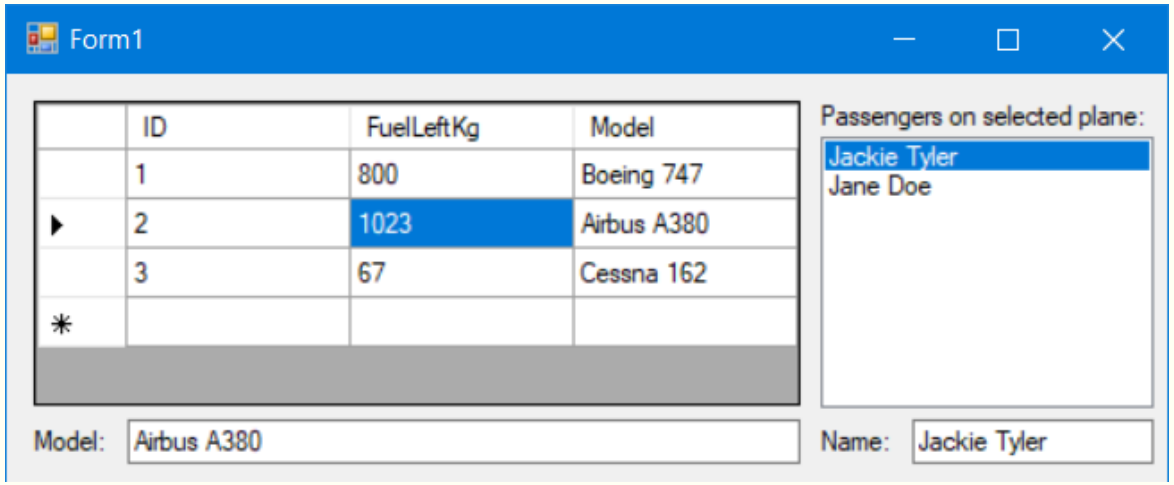



```
BindingSource bsA = new BindingSource(); // Airplanes
BindingSource bsP = new BindingSource(); // Passengers

void bsP_ListChanged(object sender, ListChangedEventArgs e)
{
    if (e.ListChangedType == ListChangedType.Reset)
        txtName.Enabled = bsP.Current != null;
}
```

Синхронізація даних між візуальними елементами

```
private void Form1_Load(object sender, EventArgs e) {
    bsP.ListChanged += new ListChangedEventHandler(bsP_ListChanged);

    // Create some example data.
    Airplane a1, a2, a3;
    bsA.Add(a1 = new Airplane("Boeing 747", 800));
    bsA.Add(a2 = new Airplane("Airbus A380", 1023));
    bsA.Add(a3 = new Airplane("Cessna 162", 67));
    a1.Passengers.Add(new Passenger("Joe Shmuck"));
    a1.Passengers.Add(new Passenger("Jack B. Nimble"));
    a1.Passengers.Add(new Passenger("Jib Jab"));
    a2.Passengers.Add(new Passenger("Jackie Tyler"));
    a2.Passengers.Add(new Passenger("Jane Doe"));
    a3.Passengers.Add(new Passenger("John Smith"));

    // Set up data binding for the parent Airplanes
    grid.DataSource = bsA;
    grid.AutoGenerateColumns = true;
    txtModel.DataBindings.Add("Text", bsA, "Model");

    // Set up data binding for the child Passengers
    bsP.DataSource = bsA; // connect the two sources
    bsP.DataMember = "Passengers";
    lstPassengers.DataSource = bsP;
    lstPassengers.DisplayMember = "Name";
    txtName.DataBindings.Add("Text", bsP, "Name");

    // Allow the user to add rows
    ((BindingList<Airplane>)bsA.List).AllowNew = true;
    ((BindingList<Airplane>)bsA.List).AllowRemove = true;
}
```





ДЯКУЮ ЗА УВАГУ!

Наступне питання: Конкурентні колекції даних