

Лабораторна робота №3

Виконав: Данильченко Олександр

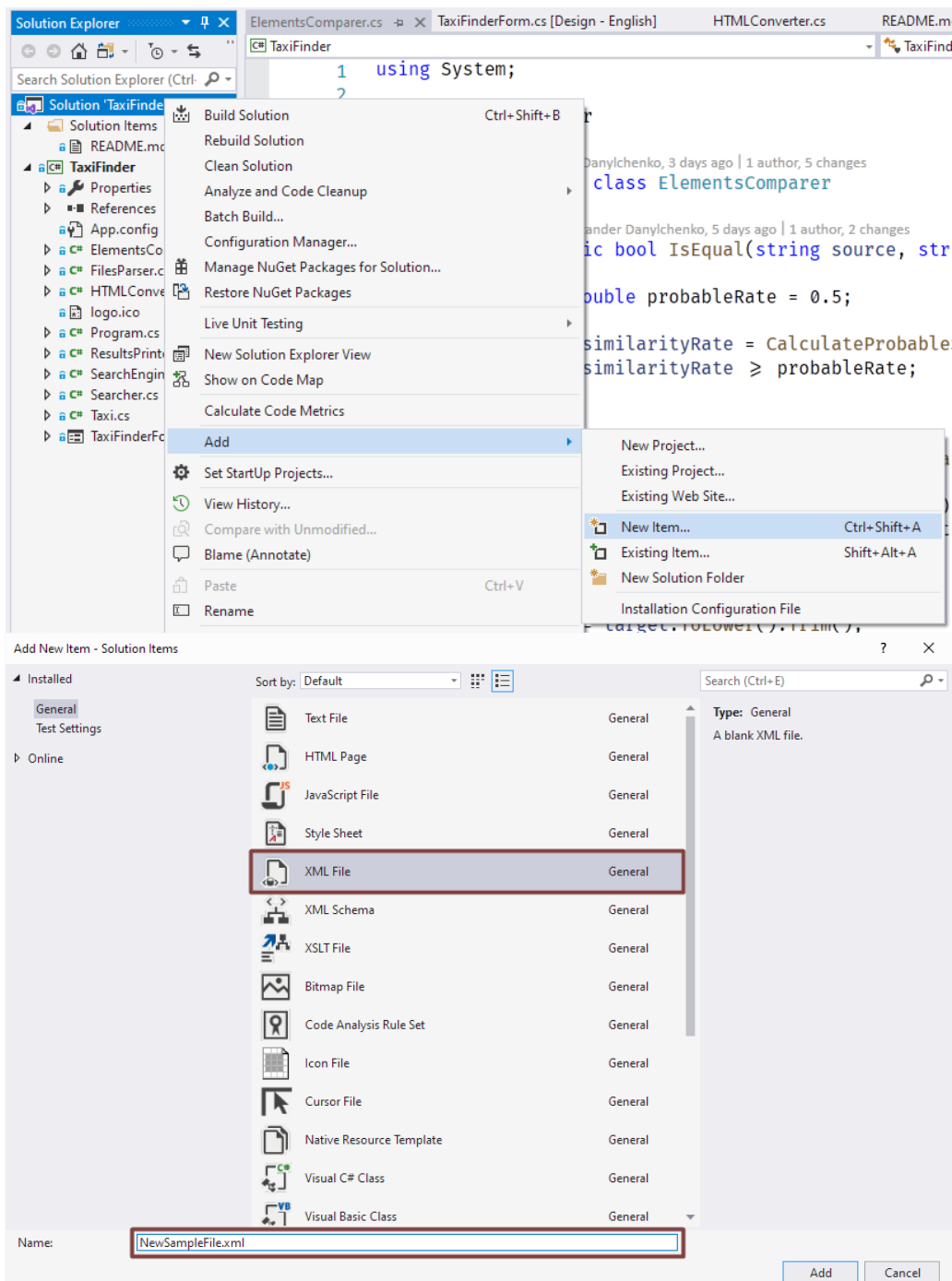
// Для зручного та продуктивного перегляду документу варто завантажити цей PDF файл,
// тому що в GDrive не працюють посилання, які допоможуть вам навігації.

Пропоную одразу поглянути на вигляд виконаної лабораторної роботи.

Етапи виконання:

1. Створення тестових XML файлів

Visual Studio дозволяє безпосередньо в IDE створювати та редагувати XML файли.



Я наполегливо рекомендую всі тестові файли зберігати у папці, до якої відбувається виведення проекту. Це пояснюється тим, що до цієї папки легко можна отримати не абсолютний, а відносний шлях завдяки засобам **C#**. Відносний шлях надає змогу працювати з даними незалежно від ПК, на якому запускається програма. У мене така папка знаходиться за адресами: `\TaxiFinder\bin\Debug\DataXML` та `\TaxiFinder\bin\Release\DataXML`, у залежності від бажаної збірки.

Приклад XML файлу:

```
<?xml version="1.0" encoding="utf-8" ?>
<Taxis>
  <Taxi>
    <Brand>Mercedes</Brand>
    <Model>AMG E43 4MATIC</Model>
    <Color>Black</Color>
    <Class>Premium</Class>
    <Driver>Kononov Vasily</Driver>
    <Number>AA5151AA</Number>
  </Taxi>
  <Taxi>
    <Brand>Mercedes</Brand>
    <Model>GLS 400D 4MATIC</Model>
    <Color>Green</Color>
    <Class>Premium</Class>
    <Driver>Pershin Anton</Driver>
    <Number>AA4251BK</Number>
  </Taxi>
  <Taxi>
    <Brand>Mercedes</Brand>
    <Model>AMG CLS 53 4MATIC+</Model>
    <Color>Silver</Color>
    <Class>Premium</Class>
    <Driver>Seman Ivan</Driver>
    <Number>AA3123MB</Number>
  </Taxi>
  <Taxi>
    <Brand>Mercedes</Brand>
    <Model>AMG GT 63S 4MATIC+</Model>
    <Color>Red</Color>
    <Class>Premium</Class>
    <Driver>Votyakova Natalia</Driver>
    <Number>AA1256MM</Number>
  </Taxi>
</Taxis>
```

Я вирішив використовувати в якості тестових даних сервіси таксі. Опорними елементами виступають автомобілі певного сервісу. За окремий сервіс відповідає файл у папці `\DataXML`. Назвам сервісів відповідають назви файлів. Вище наведено неповний перелік автомобілів служби BenzTaxi.

2. Створення менеджера вхідних файлів

Постає необхідність коректного зчитування та обробки довільної кількості XML файлів з папки `\DataXML`. Для початку варто видобути в зручні контейнери всі абсолютні шляхи до XML файлів та назви наявних сервісів.

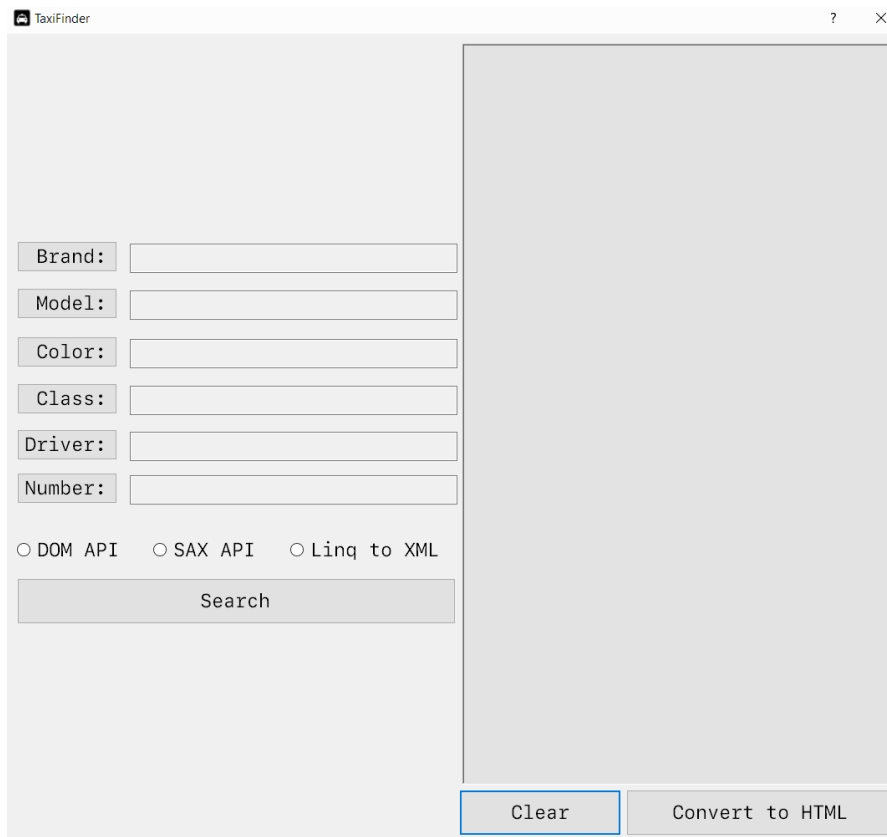
Задля цього побудуємо клас `FilesProvider` за патерном Singleton. Він добре сюди підходить, тому що потрібно зберігати єдиний екземпляр класу, що міститиме вищенаведені контейнери.

На 18ому рядку отримаємо повний шлях до папки виводу проекту. Далі до шляху виведення додається папка `\DataXML`, з якої й буде зчитано всі файли.

```
1. internal class FilesProvider
2. {
3.     private FilesProvider() { }
4.
5.     private static FilesProvider _instance;
6.     public static FilesProvider GetInstance => _instance ??
7.         (_instance = new FilesProvider()
8.         {
9.             FilesPaths = GetFilesPaths(),
10.            ServicesNames = GetServicesNames()
11.        });
12.
13.     public string[] FilesPaths { get; private set; }
14.     public string[] ServicesNames { get; private set; }
15.
16.     private static string[] GetFilesPaths()
17.     {
18.         var exePath = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
19.         var dataPath = GetDataFolderPath(exePath);
20.
21.         var filesPaths = Directory.GetFiles(dataPath, "*.xml", SearchOption.TopDirectoryOnly);
22.         return filesPaths;
23.     }
24.
25.     private static string GetDataFolderPath(string exePath)
26.     {
27.         var dataPath = string.Empty;
28.         try
29.         {
30.             dataPath = Path.Combine(exePath ?? throw new ArgumentNullException(), "DataXML");
31.         }
32.         catch (ArgumentNullException ane)
33.         {
34.             MessageBox.Show(ane.Message, ".exe path null error",
35.                 MessageBoxButtons.OK, MessageBoxIcon.Error);
36.             Application.Exit();
37.         }
38.
39.         return dataPath;
40.     }
41.
42.     private static string[] GetServicesNames()
43.     {
44.         var fileNames = GetFilesPaths()
45.             .Select(Path.GetFileNameWithoutExtension)
46.             .ToArray();
47.         return fileNames;
48.     }
49. }
```

3. Створення форми

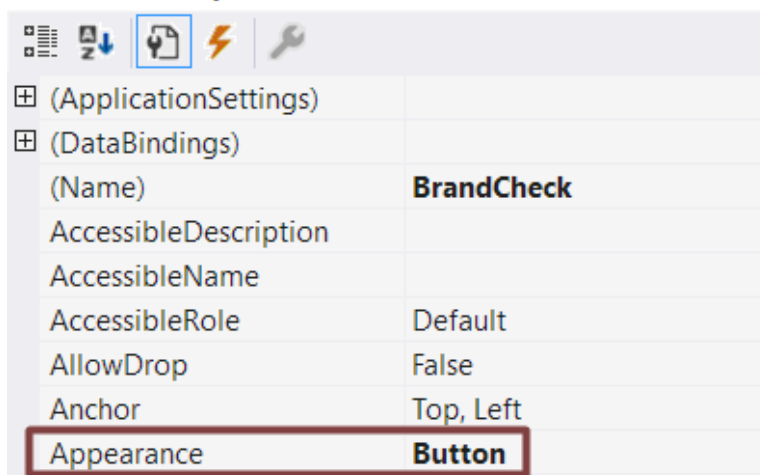
Необхідно створити front-end частину додатку, з якою й буде взаємодіяти користувач. Я обрав ось таку організацію вікна:



Тепер до особливих деталей:

- а) Для того щоб зробити **CheckBox**'и у вигляді кнопок, встановимо для кожного з них значення властивості **Appearance** як **Button**:

BrandCheck System.Windows.Forms.CheckBox

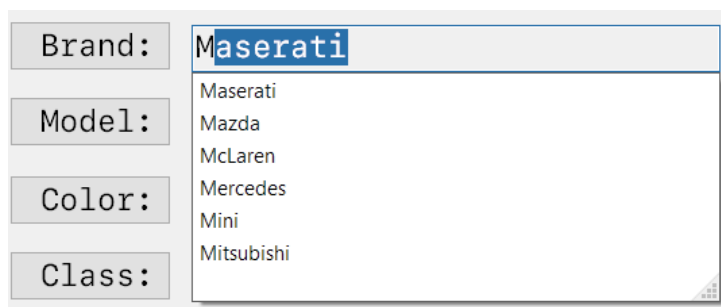
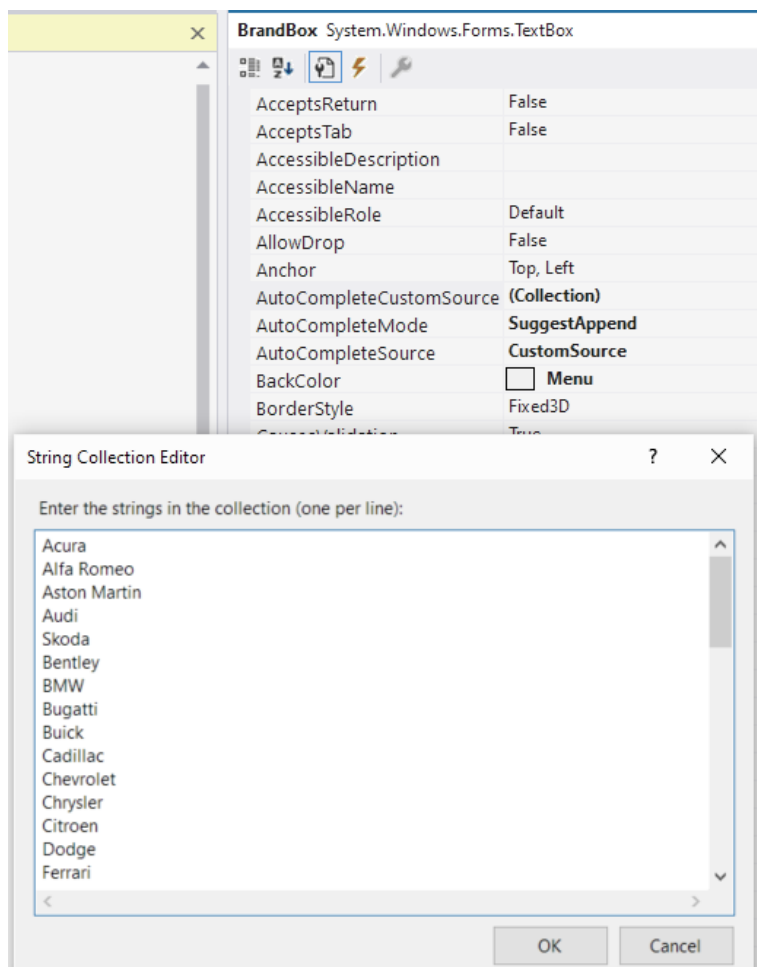


⊞ (ApplicationSettings)	
⊞ (DataBindings)	
(Name)	BrandCheck
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
Appearance	Button

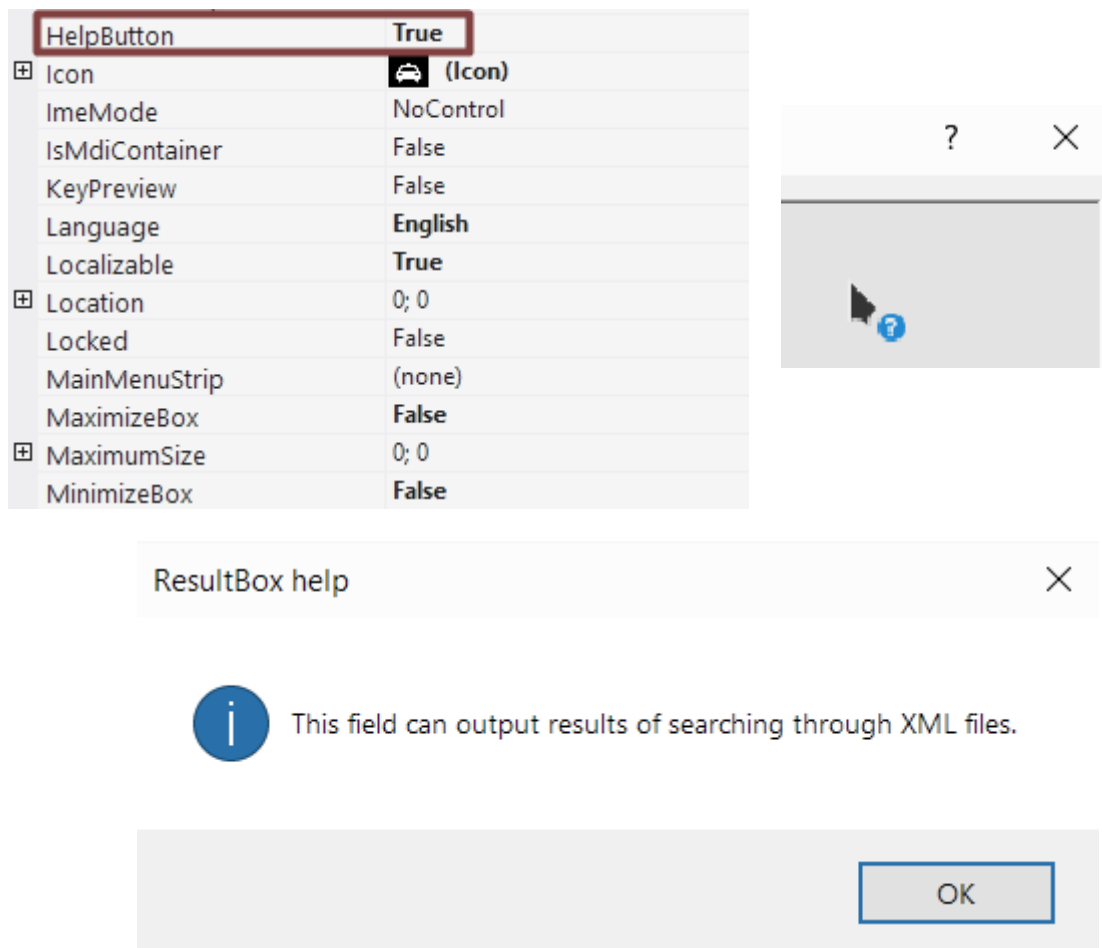
б) Для генерації запиту я використовую **TextBox**'и. Але якщо доводиться працювати з автомобілями, то кожного разу правильно вводити руками назву тієї чи іншої марки, кольору, класу чи перших двох літер державних номерів України стає надто обтяжливо. Тому довелося застосувати два дієвих прийоми. Один з них це система ймовірнісної оцінки схожості слів, про яку мова піде трохи пізніше, та автодоповнення.

Властивості **AutoComplete***** є стандартними та зручно налаштовними:

- **AutoCompleteCustomSource** це `string[]`, з якого будуть підбиратися подібні правильні слова.
- **AutoCompleteMode** визначає поведінку автодоповнювача. Я вирішив обрати спосіб при якому доповнювач як і пропонує, так і автоматично дописує вгадане слово.
- **AutoCompleteSource** визначає джерело, з якого будуть обиратися підказки. У моєму випадку воно встановлено на масив, заданий у **AutoCompleteCustomSource**



- с) Для надання довідки користувачу я використовую **HelpButton**. Цю кнопку можна увімкнути лише якщо деактивувати кнопки мінімізації та максимізації форми. Завдяки ній користувач має змогу дізнатися про будь-який об'єкт на екрані та його роль лише через один клік по ньому.



Для кожної такої відповіді у властивостях об'єктів створюємо **EventHandler** події **HelpRequested**:

HelpRequested **ResultsBox_HelpRequested**

```
1. private void ResultsBox_HelpRequested(object sender, HelpEventArgs hlpevent)
2. {
3.     MessageBox.Show("This field can output results of searching through XML files.",
4.         "ResultBox help", MessageBoxButtons.OK, MessageBoxIcon.Information);
5. }
```

4. Створення класу-контейнеру для автомобілів таксі

Цей клас являє собою контейнер з полями-властивостями, що характеризують авто в XML файлі. Цей клас знаходиться під атрибутом `[Serializable]`, тому що в подальшому ми будемо виводити `List<Taxi>` до створюваного `*.html` файлу, серіалізуючи кожен з об'єктів `Taxi`.

У ньому також є 2 методи:

- `IsAllFieldsInitialized()` – перевіряє чи всі поля мають значення.
- `IsAllFieldsBlank()` – перевіряє чи всі поля пусті.

```
1. [Serializable]
2. public class Taxi
3. {
4.     public Taxi()
5.     {
6.         Brand = Model = Color = Class = Driver = Number = string.Empty;
7.     }
8.
9.     // set property is public to allow serialization
10.    public string Brand { get; set; }
11.    public string Model { get; set; }
12.    public string Color { get; set; }
13.    public string Class { get; set; }
14.    public string Driver { get; set; }
15.    public string Number { get; set; }
16.
17.    public bool IsAllFieldsInitialized()
18.    {
19.        return Brand != string.Empty && Model != string.Empty && Color != string.Empty &&
20.            Class != string.Empty && Driver != string.Empty && Number != string.Empty;
21.    }
22.
23.    public bool IsAllFieldsBlank()
24.    {
25.        return Brand == string.Empty && Model == string.Empty && Color == string.Empty &&
26.            Class == string.Empty && Driver == string.Empty && Number == string.Empty;
27.    }
28. }
```

5. Створення класу-пошуковця

Даний клас відповідає за генерацію пошукового запиту (1), зчитування вибору пошукового рушія (2), відправлення запиту на пошук (3) та повернення списку сервісів та знайдених у них автомобілів (4).

Якщо не обрано жодного критерію пошуку чи пошукового рушія, обрані поля пусті або ж сталася помилка, то повернеться пустий список (5).

```
1. internal class Searcher
2. {
3.     private readonly TaxiFinderForm _form;
4.     public Searcher(TaxiFinderForm form)
5.     {
6.         _form = form;
7.     }
8.
9.     public List<(string service, List<Taxi> foundedTaxis)> ExecuteSearch()
10.    {
11.        var desiredTaxi = CreateSearchRequest(); (1)
12.        var engineStrategy = GetSearchEngine(); (2)
13.        if (desiredTaxi == null || engineStrategy == null)
14.        {
15.            return new List<(string, List<Taxi>>>(); (5)
16.        }
17.
18.        var executiveEngine = new SearchEngine(desiredTaxi, engineStrategy); (3)
19.
20.        try
21.        {
22.            List<(string, List<Taxi>>> results = executiveEngine.ScanAllFiles();
23.            return results; (4)
24.        }
25.        catch (NullReferenceException)
26.        {
27.            MessageBox.Show("Error occurred while reading an input XML file." +
28.                            "Try to reload data XML files.",
29.                            "XML file error", MessageBoxButtons.OK, MessageBoxIcon.Error);
30.            return new List<(string, List<Taxi>>>(); (5)
31.        }
32.    }
33.
34.    private Taxi CreateSearchRequest() (1)
35.    {
36.        var desiredTaxi = new Taxi();
37.
38.        if (_form.BrandCheck.Checked)
39.        {
40.            desiredTaxi.Brand = _form.BrandBox.Text;
41.        }
42.
43.        if (_form.ModelCheck.Checked)
44.        {
45.            desiredTaxi.Model = _form.ModelBox.Text;
46.        }
47.
48.        if (_form.ColorCheck.Checked)
49.        {
50.            desiredTaxi.Color = _form.ColorBox.Text;
51.        }
```



```
52.         if (_form.ClassCheck.Checked)
53.         {
54.             desiredTaxi.Class = _form.ClassBox.Text;
55.         }
56.         if (_form.DriverCheck.Checked)
57.         {
58.             desiredTaxi.Driver = _form.DriverBox.Text;
59.         }
60.
61.         if (_form.NumberCheck.Checked)
62.         {
63.             desiredTaxi.Number = _form.NumberBox.Text;
64.         }
65.
66.         return desiredTaxi.IsFieldsBlank() ? null : desiredTaxi;
67.     }
68.
69.     private ISearchEngineStrategy GetSearchEngine() (2)
70.     {
71.         ISearchEngineStrategy searchEngine = null;
72.
73.         if (_form.DomButton.Checked)
74.         {
75.             searchEngine = new EngineDOM();
76.         }
77.
78.         if (_form.SaxButton.Checked)
79.         {
80.             searchEngine = new EngineSAX();
81.         }
82.
83.         if (_form.LinqButton.Checked)
84.         {
85.             searchEngine = new EngineLinq();
86.         }
87.
88.         return searchEngine;
89.     }
90. }
```

6. Створення ймовірного порівнювача рядків

Це і є першим дієвим прийомом, про який я згадував раніше. Такий спосіб порівняння дозволяє користувачу робити помилки у словах чи писати їх неповністю. Програма самостійно прийме рішення про те, чи задані слова достатньо схожі щоб вважатися однаковими.

В основі цього способу лежить алгоритм обчислення «відстані Левенштейна» (1), котра рівна кількості змін, які потрібно зробити для перетворення одного виразу в інший. Далі відбувається ймовірна оцінка отриманого результату (2). Якщо слова з ймовірністю $\geq 50\%$ є одним і тим же словом, то повідомляємо про їхню рівність (3).

```
1. internal static class ElementsComparer
2. {
3.     public static bool IsEqual(string source, string target)
4.     {
5.         const double probableRate = 0.5;
6.
7.         double similarityRate = CalculateProbableSimilarity(source, target);
8.         return similarityRate >= probableRate; (3)
9.     }
10.
11.     private static double CalculateProbableSimilarity(string source, string target)
12.     {
13.         if (source == null || target == null) return 0;
14.
15.         source = source.ToLower().Trim();
16.         target = target.ToLower().Trim();
17.
18.         if (source.Length == 0 || target.Length == 0) return 0;
19.         if (source == target) return 1;
20.
21.         var stepsToSame = ComputeLevenshteinDistance(source, target); (1)
22.         var probability = 1 - (stepsToSame / (double) Math.Max(source.Length, target.Length)); (2)
23.         return probability;
24.     }
25.
26.     // Returns the number of steps required to transform the source string
27.     // into the target string.
28.     private static int ComputeLevenshteinDistance(string source, string target) (1)
29.     {
30.         // Preparations
31.         var sourceWordCount = source.Length;
32.         var targetWordCount = target.Length;
33.
34.         // Step 1
35.         if (sourceWordCount == 0)
36.         {
37.             return targetWordCount;
38.         }
39.
40.         if (targetWordCount == 0)
41.         {
42.             return sourceWordCount;
43.         }
44.
45.         // Creating measurement table
46.         var distance = new int[sourceWordCount + 1, targetWordCount + 1];
47.
```

```
48. // Step 2
49. for (var i = 0; i <= sourceWordCount;)
50. {
51.     distance[i, 0] = i++;
52. }
53.
54. for (var j = 0; j <= targetWordCount;)
55. {
56.     distance[0, j] = j++;
57. }
58.
59. for (var i = 1; i <= sourceWordCount; i++)
60. {
61.     for (var j = 1; j <= targetWordCount; j++)
62.     {
63.         // Step 3
64.         var cost = (target[j - 1] == source[i - 1]) ? 0 : 1;
65.
66.         // Step 4
67.         distance[i, j] = Math.Min(Math.Min(distance[i - 1, j] + 1, distance[i, j - 1] + 1),
68.                                     distance[i - 1, j - 1] + cost);
69.     }
70. }
71.
72. return distance[sourceWordCount, targetWordCount];
73. }
74. }
```

7. Створення трьох пошукових рушіїв

Для об'єднання трьох рушіїв у робочу структуру було обрано патерн Strategy. Пошук відбувається за запитом і рушієм, котрі були видобуті класом [Searcher](#).

У `Context`і проходимосся по всіх XML файлах, які ми отримуємо після звернення до класу [FilesProvider](#). Ці файли по черзі передаються на опрацювання встановленому рушію. Після обробки знайдені автомобілі додаємо до результуючого списку.

```
1. // Context
2. internal class SearchEngine
3. {
4.     private readonly Taxi _desiredTaxi;
5.     private readonly ISearchEngineStrategy _engine;
6.
7.     public SearchEngine(Taxi desiredTaxi, ISearchEngineStrategy engine)
8.     {
9.         _desiredTaxi = desiredTaxi;
10.        _engine = engine;
11.    }
12.
13.    public List<(string service, List<Taxi> foundedTaxis)> ScanAllFiles()
14.    {
15.        var results = new List<(string, List<Taxi>>>();
16.
17.        var filePaths = FilesProvider.GetInstance.FilePaths;
18.        var servicesNames = FilesProvider.GetInstance.ServicesNames;
19.
20.        for (var i = 0; i < filePaths.Length; ++i)
21.        {
22.            var serviceName = servicesNames[i];
23.            var filePath = filePaths[i];
24.            var foundedTaxisInService = _engine.DoSearchInFile(filePath, _desiredTaxi);
25.
26.            results.Add((serviceName, foundedTaxisInService));
27.        }
28.
29.        return results;
30.    }
31. }
```

У Strategy фіксуємо загальний метод обробки:

```
1. // Strategy
2. internal interface ISearchEngineStrategy
3. {
4.     List<Taxi> DoSearchInFile(string filePath, Taxi desiredTaxi);
5. }
```

У пошукових рушіях скористаємося вже написаним [ймовірнісним порівнювачем](#). Але використання обмежимо лише полями **Brand**, **Color**, **Class** та **Driver**. Це пов'язано з тим, що визначення автомобіля за моделлю та номером потребує суттєво більшої точності. Це зводить нанівець потребу у використанні додаткового алгоритму та його модифікації, тому що потрібна функціональність вже реалізована в методі `.Contains()`.

Реалізації пошукових рушіїв на DOM API (1), SAX API (2) та Linq To XML (3):

```

1. // Concrete Strategy A
2. internal class EngineDOM : ISearchEngineStrategy (1)
3. {
4.     public List<Taxi> DoSearchInFile(string filePath, Taxi desiredTaxi)
5.     {
6.         var foundedTaxis = new List<Taxi>();
7.
8.         var xDoc = new XmlDocument();
9.         xDoc.Load(filePath);
10.        var xRoot = xDoc.DocumentElement;
11.
12.        var taxiNodes = xRoot?.SelectNodes("Taxi");
13.        if (taxiNodes != null)
14.        {
15.            foreach (XmlElement taxi in taxiNodes)
16.            {
17.                var newTaxi = new Taxi();
18.
19.                foreach (XmlElement element in taxi)
20.                {
21.                    if (element.Name == "Brand" && (string.IsNullOrEmpty(desiredTaxi.Brand) ||
22.                                                    ElementsComparer.IsEqual(element.InnerText,
23.                                                                desiredTaxi.Brand)))
24.                    {
25.                        newTaxi.Brand = element.InnerText;
26.                    }
27.
28.                    else if (element.Name == "Model" && (string.IsNullOrEmpty(desiredTaxi.Model) ||
29.                                                         element.InnerText.ToUpper()
30.                                                         .Contains(desiredTaxi.Model.ToUpper()))))
31.                    {
32.                        newTaxi.Model = element.InnerText;
33.                    }
34.
35.                    else if (element.Name == "Color" && (string.IsNullOrEmpty(desiredTaxi.Color) ||
36.                                                         ElementsComparer.IsEqual(element.InnerText,
37.                                                                desiredTaxi.Color)))
38.                    {
39.                        newTaxi.Color = element.InnerText;
40.                    }
41.
42.                    else if (element.Name == "Class" && (string.IsNullOrEmpty(desiredTaxi.Class) ||
43.                                                         ElementsComparer.IsEqual(element.InnerText,
44.                                                                desiredTaxi.Class)))
45.                    {
46.                        newTaxi.Class = element.InnerText;
47.                    }
48.
49.                    else if (element.Name == "Driver" && (string.IsNullOrEmpty(desiredTaxi.Driver)
50.                                                         || ElementsComparer.IsEqual(element.InnerText,
51.                                                                desiredTaxi.Driver)))
52.                    {
53.                        newTaxi.Driver = element.InnerText;
54.                    }
55.
56.                    else if (element.Name == "Number" && (string.IsNullOrEmpty(desiredTaxi.Number)
57.                                                         || element.InnerText.ToUpper()
58.                                                         .Contains(desiredTaxi.Number.ToUpper()))))
59.                    {
60.                        newTaxi.Number = element.InnerText;
61.                    }
62.                }
63.
64.                if (newTaxi.IsAllFieldsInitialized())
65.                {
66.                    foundedTaxis.Add(newTaxi);
67.                }
68.            }
69.        }
70.
71.        return foundedTaxis;
72.    }
73. }

```

```

1.  // Concrete Strategy B
2.  internal class EngineSAX : ISearchEngineStrategy (2)
3.  {
4.      public List<Taxi> DoSearchInFile(string filePath, Taxi desiredTaxi)
5.      {
6.          var foundedTaxis = new List<Taxi>();
7.          using (var xr = XmlReader.Create(filePath))
8.          {
9.              var iteratorTaxi = new Taxi();
10.
11.              var element = string.Empty;
12.              while (xr.Read())
13.              {
14.                  // Reads the element
15.                  if (xr.NodeType == XmlNodeType.Element)
16.                  {
17.                      element = xr.Name; // The name of the current element
18.                  }
19.                  // Reads the element value
20.                  else if (xr.NodeType == XmlNodeType.Text)
21.                  {
22.                      if (element == "Brand" && (string.IsNullOrEmpty(desiredTaxi.Brand) ||
23.                          ElementsComparer.IsEqual(xr.Value, desiredTaxi.Brand)))
24.                      {
25.                          iteratorTaxi.Brand = xr.Value;
26.                      }
27.                      else if (element == "Model" && (string.IsNullOrEmpty(desiredTaxi.Model) ||
28.                          xr.Value.ToUpper()
29.                              .Contains(desiredTaxi.Model.ToUpper())))
30.                      {
31.                          iteratorTaxi.Model = xr.Value;
32.                      }
33.
34.                      else if (element == "Color" && (string.IsNullOrEmpty(desiredTaxi.Color) ||
35.                          ElementsComparer.IsEqual(xr.Value, desiredTaxi.Color)))
36.                      {
37.                          iteratorTaxi.Color = xr.Value;
38.                      }
39.
40.                      else if (element == "Class" && (string.IsNullOrEmpty(desiredTaxi.Class) ||
41.                          ElementsComparer.IsEqual(xr.Value, desiredTaxi.Class)))
42.                      {
43.                          iteratorTaxi.Class = xr.Value;
44.                      }
45.
46.                      else if (element == "Driver" && (string.IsNullOrEmpty(desiredTaxi.Driver) ||
47.                          ElementsComparer.IsEqual(xr.Value, desiredTaxi.Driver)))
48.                      {
49.                          iteratorTaxi.Driver = xr.Value;
50.                      }
51.
52.                      else if (element == "Number" && (string.IsNullOrEmpty(desiredTaxi.Number) ||
53.                          xr.Value.ToUpper()
54.                              .Contains(desiredTaxi.Number.ToUpper())))
55.                      {
56.                          iteratorTaxi.Number = xr.Value;
57.                      }
58.                  }
59.
60.                  // Reads the closing element
61.                  else if ((xr.NodeType == XmlNodeType.EndElement) && (xr.Name == "Taxi"))
62.                  {
63.                      if (iteratorTaxi.IsAllFieldsInitialized())
64.                      {
65.                          var newTaxi = iteratorTaxi;
66.                          foundedTaxis.Add(newTaxi);
67.                          iteratorTaxi = new Taxi();
68.                      }
69.                  }
70.              }
71.          }
72.          return foundedTaxis;
73.      }
74.  }

```

```

1. // Concrete Strategy C
2. internal class EngineLinq : ISearchEngineStrategy (3)
3. {
4.     public List<Taxi> DoSearchInFile(string filePath, Taxi desiredTaxi)
5.     {
6.         var xdoc = XDocument.Load(filePath);
7.         var foundedElements = from elem in xdoc.Element("Taxis").Elements("Taxi")
8.             where
9.                 (string.IsNullOrEmpty(desiredTaxi.Brand) ||
10.                  ElementsComparer.IsEqual(elem.Element("Brand").Value, desiredTaxi.Brand)) &&
11.
12.                 (string.IsNullOrEmpty(desiredTaxi.Model) ||
13.                  elem.Element("Model").Value.ToUpper().Contains(desiredTaxi.Model.ToUpper())) &&
14.
15.                 (string.IsNullOrEmpty(desiredTaxi.Color) ||
16.                  ElementsComparer.IsEqual(elem.Element("Color").Value, desiredTaxi.Color)) &&
17.
18.                 (string.IsNullOrEmpty(desiredTaxi.Class) ||
19.                  ElementsComparer.IsEqual(elem.Element("Class").Value, desiredTaxi.Class)) &&
20.
21.                 (string.IsNullOrEmpty(desiredTaxi.Driver) ||
22.                  ElementsComparer.IsEqual(elem.Element("Driver").Value, desiredTaxi.Driver)) &&
23.
24.                 (string.IsNullOrEmpty(desiredTaxi.Number) ||
25.                  ElementsComparer.IsEqual(elem.Element("Number").Value, desiredTaxi.Number))
26.
27.         select new Taxi
28.         {
29.             Brand = elem.Element("Brand").Value.ToString(),
30.             Model = elem.Element("Model").Value.ToString(),
31.             Color = elem.Element("Color").Value.ToString(),
32.             Class = elem.Element("Class").Value.ToString(),
33.             Driver = elem.Element("Driver").Value.ToString(),
34.             Number = elem.Element("Number").Value.ToString()
35.         };
36.
37.         return foundedElements.ToList();
38.     }
39. }

```

8. Створення класу-друкаря

Створимо невеликий клас для форматування та виведення результатів пошуку в заданий через конструктор класу `RichTextBox`.

```
1.  internal class ResultsPrinter
2.  {
3.      private readonly RichTextBox _outputBox;
4.      public ResultsPrinter(RichTextBox outputBox)
5.      {
6.          _outputBox = outputBox;
7.      }
8.
9.      public void Print(IEnumerable<(string service, List<Taxi> foundedTaxis)> results)
10.     {
11.         _outputBox.Text = string.Empty;
12.         foreach (var (service, foundedTaxis) in results)
13.         {
14.             _outputBox.AppendText($"\\n{service}:\\n");
15.
16.             var outputNumber = 1;
17.             foreach (var taxiOutput in foundedTaxis
18.                 .Select(taxi => $"{{outputNumber}}:\\n" +
19.                    $"  Brand: {{taxi.Brand}}\\n" +
20.                    $"  Model: {{taxi.Model}}\\n" +
21.                    $"  Color: {{taxi.Color}}\\n" +
22.                    $"  Class: {{taxi.Class}}\\n" +
23.                    $"  Driver: {{taxi.Driver}}\\n" +
24.                    $"  Number: {{taxi.Number}}\\n"))
25.             {
26.                 _outputBox.AppendText(taxiOutput);
27.                 ++outputNumber;
28.             }
29.         }
30.     }
31. }
```


9. Створення XSL файлу

Для трансформації необхідно створити *.xsl файл-шаблон, який задає правила розмітки та розшифрування елементів вхідного *.xml файлу у вихідний *.html файл.

Результуючий файл уже буде являти собою готову веб-сторінку, яку можна запустити у будь-якому браузері.

Для більшої зручності, надійності та швидкості рекомендую скомпілювати *.xsl файл у *.dll формат та підключити його до свого проекту через **References**. Про це варто детальніше прочитати та знайти покрокову інструкцію по використанню за посиланням:

<https://docs.microsoft.com/en-us/dotnet/standard/data/xml/how-to-perform-an-xslt-transformation-by-using-an-assembly>.

Ось так виглядає мій *.xsl файл:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/Taxis">
    <html>
      <head>
        <title>Search results:</title>
        <link rel="shortcut icon" href="http://www.iconj.com/ico/t/j/tjfrfeyrty.ico" type="image/x-icon" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
      </head>
      <style>
        .content-table {
          border-collapse: collapse;
          margin: 25px 0;
          border-spacing: 5px;
          font-size: 0.9em;
          min-width: 420px;
          border-radius: 7px 7px 0 0;
          overflow: hidden;
          box-shadow: 0 0 20px rgba(0, 0, 0, 0.17);
        }

        .content-table thead tr {
          background: #F5AE1C;
          text-align: left;
          font-weight: bold;
        }

        .content-table th,
        .content-table td {
          padding: 12px 15px;
        }

        .content-table tbody tr {
          border-bottom: 1px solid #DCDCDC;
        }

        .content-table tbody tr:nth-of-type(even) {
          background-color: #F3F3F3;
        }

        .content-table tbody tr:last-of-type {
          border-bottom: 2px solid #F5AE1C;
        }

        body {
          font-family: SF Mono !important;
          font-size: 28px;
        }
      </style>
    </html>
  </template>
</xsl:stylesheet>
```

```

</head>

<body>
<h1>Search results:</h1>
<table class="content-table">
  <thead>
    <tr>
      <th>Brand</th>
      <th>Model</th>
      <th>Color</th>
      <th>Class</th>
      <th>Driver</th>
      <th>Number</th>
    </tr>
  </thead>
  <xsl:for-each select="Taxi">
    <tr>
      <td>
        <xsl:value-of select="Brand" />
      </td>
      <td>
        <xsl:value-of select="Model" />
      </td>
      <td>
        <xsl:value-of select="Color" />
      </td>
      <td>
        <xsl:value-of select="Class" />
      </td>
      <td>
        <xsl:value-of select="Driver" />
      </td>
      <td>
        <xsl:value-of select="Number" />
      </td>
    </tr>
  </xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

Результатом такого перетворення є красива таблиця:

Search results:

Brand	Model	Color	Class	Driver	Number
Mercedes	AMG E43 4MATIC	Black	Premium	Kononov Vasily	AA5151AA
Mercedes	AMG CLS 53 4MATIC+	Silver	Premium	Seman Ivan	AA3123MB
Mercedes	AMG GT 63S 4MATIC+	Red	Premium	Votyakova Natalia	AA1256MM
Mercedes	AMG SLC 43	Blue	Premium	Chernukha Andrey	AA3333PL

10. Створення класу-перетворювача

Я вирішив перетворювати в *.html файл результати здійсненого пошуку.

Процес перетворення розпочинається лише якщо список знайдених авто не пустий. Також користувач через стандартне діалогове вікно сам може вибирати назву файлу та подальше місце його зберігання (1).

Безпосередньо перетворення відбувається в 3 етапи:

1. Створення тимчасового *.xml файлу. У нього буде серіалізовано список знайдених авто (List<Taxi>) (2).
2. Перетворення тимчасового *.xml файлу в *.html файл за попередньо скомпільованим *.xsl шаблоном (3).
3. Видалення тимчасового *.xml файлу (4).

```
1. internal class HTMLConverter
2. {
3.     private readonly List<Taxi> _outputTaxis;
4.     private readonly TaxiFinderForm _form;
5.
6.     public HTMLConverter(IEnumerable<(string service, List<Taxi> foundedTaxis)> results,
7.                                     TaxiFinderForm form)
8.     {
9.         _outputTaxis = new List<Taxi>();
10.        foreach (var (_, foundedTaxis) in results)
11.        {
12.            _outputTaxis.AddRange(foundedTaxis);
13.        }
14.        _form = form;
15.    }
16.
17.    public void Convert()
18.    {
19.        if (_outputTaxis.Count != 0 && _form.HTMLSaveDialog.ShowDialog() == DialogResult.OK) (1)
20.        {
21.            var htmlPath = _form.HTMLSaveDialog.FileName;
22.            var xmlPath = ExtractResultsInTempXML(htmlPath); (2)
23.
24.            TransformWithTemplate(xmlPath, htmlPath); (3)
25.            RemoveTempXML(xmlPath); (4)
26.        }
27.    }
28.
29.    private string ExtractResultsInTempXML(string htmlPath) (2)
30.    {
31.        var xmlFilePath = htmlPath.Replace(".html", ".xml");
32.
33.        var fs = new FileStream(xmlFilePath, FileMode.Create);
34.        var serializer = new XmlSerializer(_outputTaxis.GetType(), new XmlRootAttribute("Taxis"));
35.        serializer.Serialize(fs, _outputTaxis);
36.        fs.Close();
37.
38.        return xmlFilePath;
39.    }
40.
```

```
41. private static void TransformWithTemplate(string xmlPath, string htmlPath) (3)
42. {
43.     var xslt = new XslCompiledTransform();
44.     xslt.Load(typeof(HTMLTransform));
45.     xslt.Transform(xmlPath, htmlPath);
46. }
47.
48. private static void RemoveTempXML(string xmlPath) (4)
49. {
50.     File.Delete(xmlPath);
51. }
52. }
```

11. Під'єднання реалізованої функціональності до кнопок форми

Для зберігання отриманих результатів у `TaxiFinderForm` створимо список результатів. У ньому зберігається пара: «сервіс – список знайдених авто» (1).

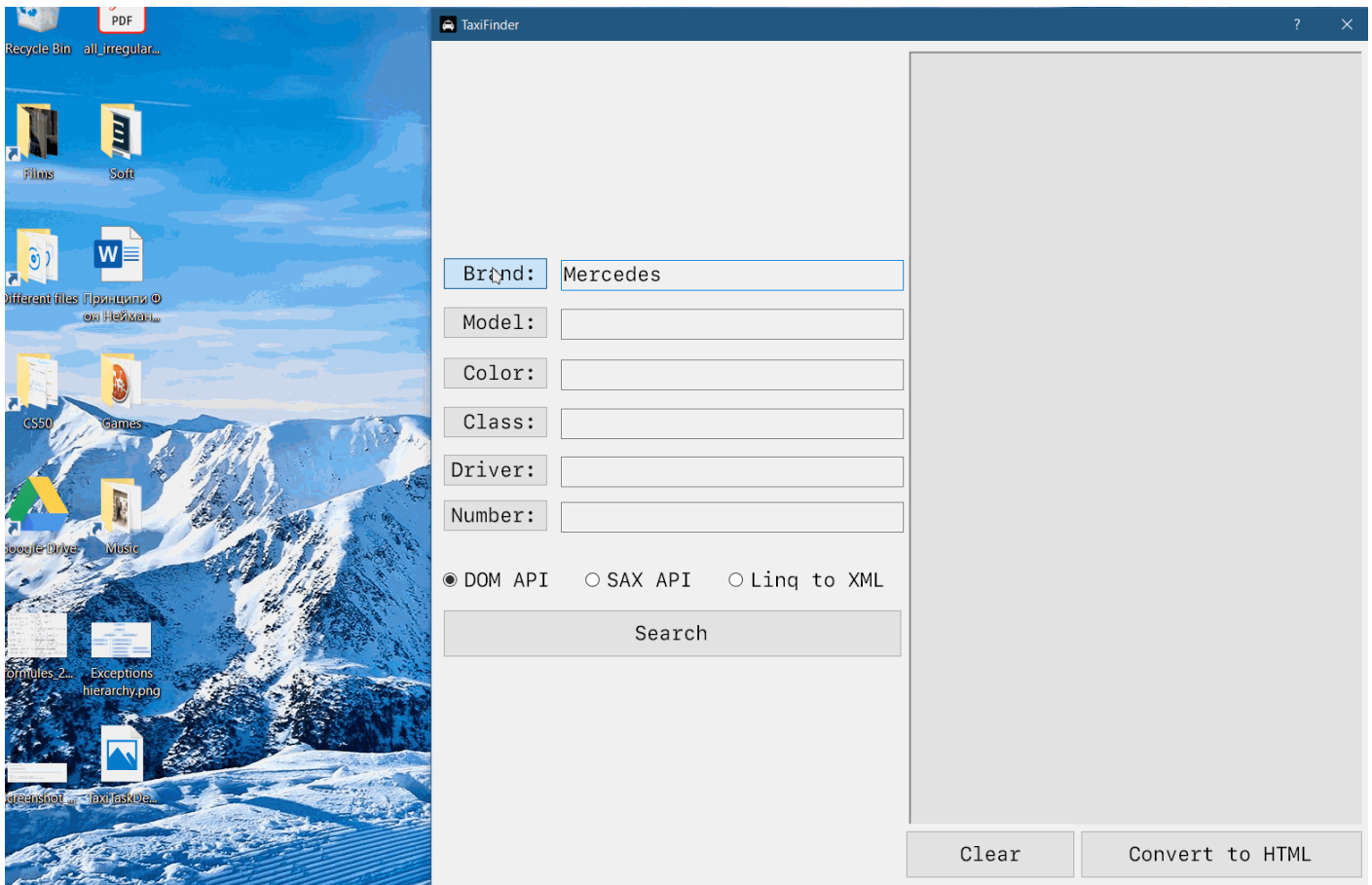
- Натискання на кнопку **Search** виконує 2 прості операції однакового рівня абстракції. Таким чином уникаємо анти-патерну **Magic Button**.
Спочатку ініціюємо пошук через метод `ExecuteSearch()`. Після успішного, або не дуже, пошуку ініціюється виведення інформації на екран через метод `.Print(IEnumerable...)`. (2)
- Натискання на кнопку **Clear** усього-на-всього очищає `RichTextBox` для виведення інформації. (3)
- Натискання на кнопку **Convert to HTML** ініціює перетворення виведених на екран результатів у `*.html` файл через метод `.Convert()`. (4)

```
1. public partial class TaxiFinderForm : Form
2. {
3.     private List<(string service, List<Taxi> foundedTaxis)> _results =
                                                new List<(string, List<Taxi>)>(); (1)
4.
5.     public TaxiFinderForm()
6.     {
7.         InitializeComponent();
8.     }
9.
10.    private void SearchButton_Click(object sender, EventArgs e) (2)
11.    {
12.        var searcher = new Searcher(this);
13.        _results = searcher.ExecuteSearch();
14.
15.        var printer = new ResultsPrinter(ResultsBox);
16.        printer.Print(_results);
17.    }
18.
19.    private void ClearButton_Click(object sender, EventArgs e) (3)
20.    {
21.        ResultsBox.Text = string.Empty;
22.        _results.Clear();
23.    }
24.
25.    private void ConvertToHTMLButton_Click(object sender, EventArgs e) (4)
26.    {
27.        var converter = new HTMLConverter(_results, this);
28.        converter.Convert();
29.    }
30.
31.    // далі йде перелік методів-відповідей ...
32.
33. }
```

Повний код `*.cs` файлу форми: [TaxiFinderForm.cs](#).

12. Насолодження результатом нашої кропіткої тривалої праці

// Для перегляду демо натисніть на картинку



Recycle Bin all_irregular...

PDF

Files

Soft

different files Принципи 0 на Нейман...

CS50

Games

Google Drive

Music

formules.2... Exceptions hierarchy.png

taxiaskoda

TaxiFinder

Brand: Mercedes

Model:

Color:

Class:

Driver:

Number:

☒ DOM API ☐ SAX API ☐ Linq to XML

Search

Clear

Convert to HTML

Search results:

Brand	Model	Color	Class	Driver	Number
Skoda	Fabia	Red	Standart	Krapinevich Alexander	AA1257TT