Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра Програмної інженерії

КУРСОВА РОБОТА

ПОЯСНЮВАЛЬНА ЗАПИСКА

з дисципліни “ Об’єктно - орієнтоване програмування”

“Довідник меломана”

Керівник , проф. Бондарєв В.М.

Студент гр. ПІ - 15 – 1 Деревнін О.О.

Комісія:

проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дудар З.В.

<посада> \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_<прізвище, ініціали >

<посада> \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_<прізвище, ініціали >

Харків 2016

**ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ**

Кафедра: ***Програмної інженерії***

Дисципліна: ***Об’єктно-орієнтоване програмування***

Спеціальність: ***Програмна інженерія***

Курс 1 .Група *ПІ-15-1* . Семестр *2* .

***ЗАВДАННЯ***

***на курсовий проект студента***

*Деревніна Олександра Олександровича*

1. Тема проекту: «Довідник меломана»
2. Термін здачі студентом закінченого проекту: ***“01” - червня - 2016 р***.
3. Вихідні дані до проекту: база груп та виконавців, база пісень, база дисків з переліком пісень (у вигляді посилань). Вибір усіх пісень заданої групи, усіх дисків, де є задана пісня.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |
| --- | --- | --- |
| *№* | *Назва етапу* | *Термін виконання* |
| 1 | Видача теми, узгодження і затвердження теми | 1-03-2016 р. |
| 2 | Формулювання вимог до програми | 1-03-2016 – 6-03-2016 р. |
| 3 | Розробка моделі програми | 07-03-2016 – 28-03-2016 р. |
| 4 | Розробка дизайну програми | 01-04-2016 – 18-04-2016 р. |
| 5 | Розробка підсистеми додавання пісень, дисків, виконавців та їх зберігання. | 19-04-2016 – 31-04-2016 р. |
| 6 | Тестування і доопрацювання розробленої програмної системи. | 01-05-2016 – 20-05-2016 р. |
| 7 | Оформлення пояснювальної записки, додатків, графічного матеріалу | 21-05-2016 – 31-06-2016 р. |
| 8 | Захист | 01-06-2016 р. |

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Деревнін О.О.

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Бондарєв В. М.

« 1 »\_березня\_\_\_\_\_\_\_\_\_\_ 2016 р.

РЕФЕРАТ

Пояснювальна записка до курсової роботи: 45 с., 20 Рисунків, 1 додаток, 4 джерела.

Метою роботи є розробка програми “Довідник меломана” на засадах об'єктно-орієнтованого програмування.

Методи розробки базуються на використанні середи розробки Microsoft Visual Studio 2015 Сommunity, Windows Forms, платформи .NET Framework 4.5, мови програмування C#.

Профіль на github: https://github.com/Deryalen/

В результаті отримана програма під назвою “Довідник меломана”, яка дозволяє додавати пісні, альбоми та виконавців до бази, виконувати пошук за ними, зберігати дані до файлу. Пісні характеризуються за назвою та датою публікації. Альбоми – за назвою, датою публікації та списком пісень. Виконавці – за іменем чи назвою гурту, датою народження чи заснування гурту та списком альбомів.

ОБ’ЕКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ, ПРОГРАМА, КЛАС, C#, ПЛАТФОРМА .NET, ПІСНІ, АЛЬБОМИ, ДИСКИ, ВИКОНАВЦІ, ГРУПИ

ЗМІСТ

Вступ 6

розділ 1. Специфікація програми 7

1.1 Функції програми 7

1.2 Інтерфейс користувача 8

розділ 2. Проектна специфікація. 15

2.1 Архітектурний стиль 15

2.2. Об’єктна структура програми 15

2.3 Формат даних 16

розділ 3. Інструкція користувача 17

ВИСНОВКИ 20

ПЕРЕЛІК ПОСИЛАНЬ.. 21

Додаток А – Код програми.. 22

ВСТУП

У наш час більшість людей полюбляє слухати музику. Це недорогий та простий спосіб підняти собі настрій, чи згадати приємні моменти зі свого життя.

Але досить часто трапляється так, що людина забуває назву пісні, чи не може згадати, хто її виконує.

Дослідивши переваги та недоліки паперових носіїв інформації для зберігання даних про пісні, альбоми та виконавців, будо прийнято рішення створити прикладну програму, щоб вирішити проблеми, перелічені вище.

З метою полегшення пошуку та структурування пісень, альбомів та виконавців, була розроблена програма, що виконує вказані функції.

Дизайн програми близький до стилю «New Retro Wave», який став популярним останнім часом і стосується як музики, так і мистецтва взагалі, а мінімалізм дозволить користувачу не відволікатися при пошуку.

РОЗДІЛ 1. Специфікація програми

1.1 Функції програми

Програма повинна дозволяти користувачу зберігати дані пісень, альбомів та виконавців.

Програма, що буде розроблена, повинна мати наступні функції:

1. відображення баз пісень, альбомів, виконавців;
2. додавання нових елементів до баз;
3. видалення елементів з баз;
4. пошук елементів за назвою;
5. відображення пісень у вибраному альбомі у вигляді посилань

1.3 Інтерфейс користувача

Після запуску програми користувач побачить головну панель, вона ж панель відображення, пошуку та управління базою меломана (Рисунок 1.1). 

Рисунок 1.1

При першому натисканні на бажаний пункт меню, відображається повідомлення про відсутність даних та коРисуноктовачу запропоновується додати інформацію до бази (Рисунок 1.2).

Рисунок 1.2

При натисканні коРисуноктувачем на кнопку додавання, відкривається форма для додавання елемента (пісні (Рисунок 1.3), альбому (за наявності пісень) (Рисунок1.4), виконавця (за наявності пісень чи альбомів) (Рисунок 1.5)

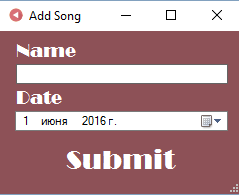


Рисунок 1.3

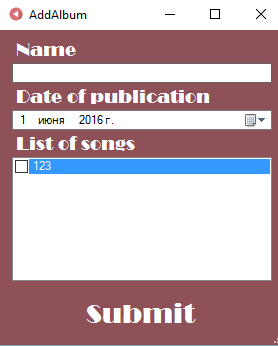


Рисунок 1.4



Рисунок 1.5

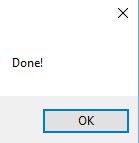
При введенні даних та натисканні кнопки «Submit» користувач побачить повідомлення (Рисунок 1.6)

Рисунок 1.6

У подальшому, при натисканні на пункт меню з назвою елемента, який був доданий, у головній формі буде показано список елементів (Рисунок 1.7).

При натисканні правою кнопкою миші на рядок з елементом буде показано контекстне меню, яке дозволяє відкрити форму редагування чи видалити елемент (Рисунок 1.8).

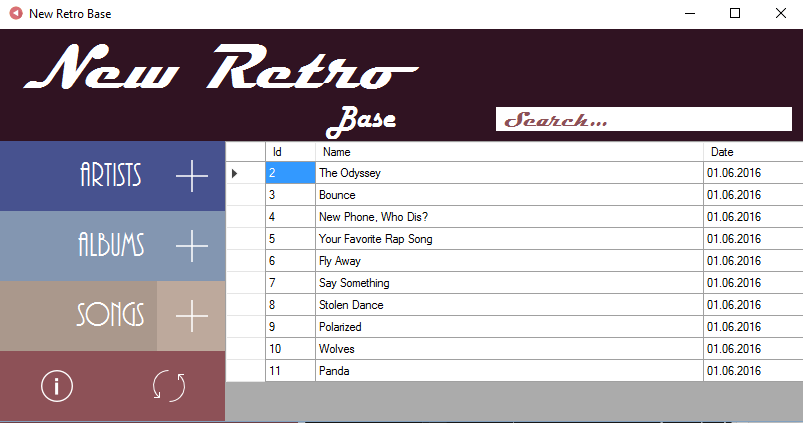


Рисунок 1.7



Рисунок 1.8

При натисканні на кнопку «Delete» елемент буде видалено з бази, а список оновлено. Користувач побачить повідомлення (Рисунок 1.9)

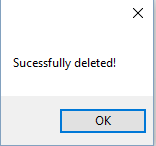


Рисунок 1.9

При натисканні на кнопку «Show info or edit» буде відкрита форма з детальною інформацією про елемент та можливістю його редагування при натисканні на кнопку «Edit» (Рисунок 1.10, Рисунок 1.11, Рисунок 1.12)

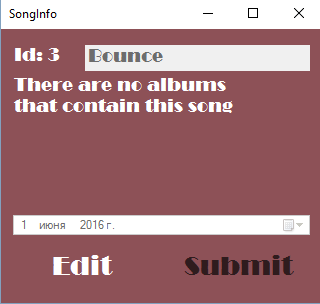


Рисунок 1.10



Рисунок 1.11

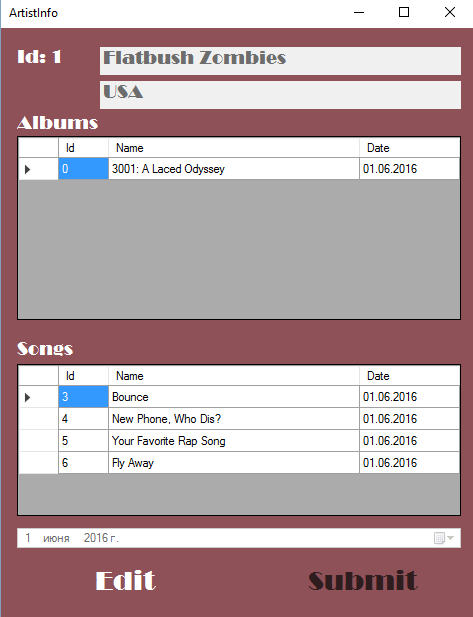


Рисунок 1.12

При подвійному натисканні на будь-який елемент у списках пісень чи альбомів у меню з інформацією про альбом чи виконавця, буде відкрито аналогічну форму з інформацією про вибраний елемент.

При натисканні правою кнопкою миші буде показано аналогічне контекстне меню (Рисунок 1.8).

При натисканні на кнопку з зображенням «і» буде викликано меню з інформацією про програму (Рисунок 1.13)

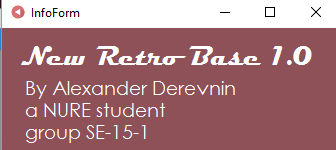


Рисунок 1.13

При натисканні на кнопку з зображенням стрілок списки елементів буде оновлено.

При введенні пошукового запиту у поле «Search», у активному списку буде відображено лише ті елементи, які відповідають критеріям пошуку. Пошук ведеться за всіма стовпцями одразу.

РОЗДІЛ 2. Проектна специфікація

2.1 Архітектурний стиль

Для виконання даного проекту була використана видозмінена архітектура MVC. У нашому випадку, логіка розділена на два компоненти: Model, View/Controller.

Model – частина програми, в якій знаходиться вся функціональна логіка програми.

View/Controller – через використання Windows Forms, вдалося об’єднати ці два компоненти, які відповідають за зображення моделі, реагування на зовнішні події та змінення моделі та зображення.

Завдяки використанню MVC, можна змінювати відображення, не змінюючи модель, що є важливою та зручною властивістю.

2.2 Об’єктна структура програми

Програма представляє собою набір класів, взаємодіючих між собою

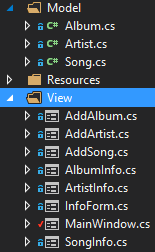
(Рисунок 2.1).

Рисунок 2.1

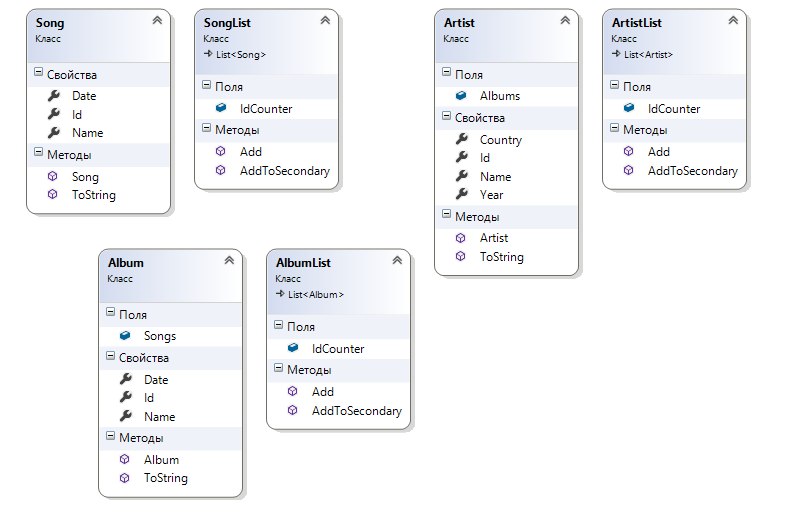
 У програмі реалізовані наступні класи: Album, Artist, Song, AlbumList, ArtistList, SongList (Рисунок 2.2)

Рисунок 2.2

2.3 Формат даних

Для зберігання інформації використовується серіалізація до.bin-файлу (бінарна серіалізація), яка дозволяє зберігати об’єкти та захищає базу від ручної зміни користувачем.

РОЗДІЛ 3. Інструкція користувача

Встановлення програми:

* Відкрити «Мій комп’ютер»
* Знайти носій з програмою
* Розпакувати архів у бажану папку на комп’ютері
* Запустити .exe-файл та розпочати роботу з програмою

Робота з програмою:

Запустити програму. Після відкриття вікна з програмою додати бажані

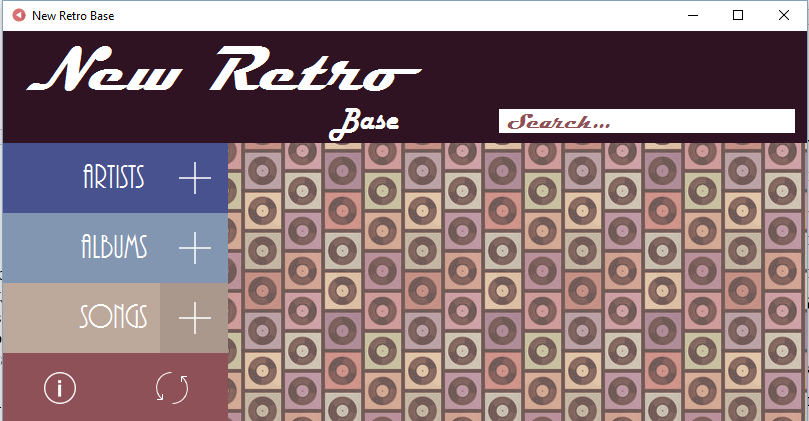
елементи до бази, користуючись кнопками додавання поруч з елементами графічного меню (Рисунок 3.1)

Рисунок 3.1

У формі, що з’явилася, заповнити всі текстові поля, вибрати дату публікації, народження або заснування гурту, вибрати у списку пісні, що входять до альбому або пісні та альбоми, що належать виконавцю (Рисунок 3.2, Рисунок 3.3, Рисунок 3.4)

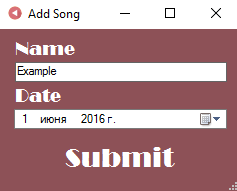


Рисунок 3.2

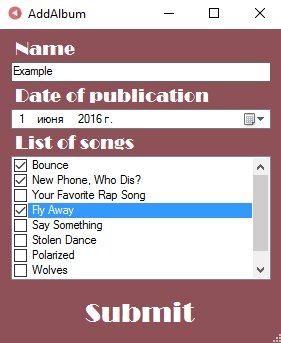


Рисунок 3.3

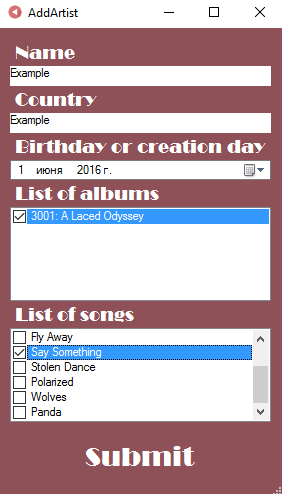


Рисунок 3.4

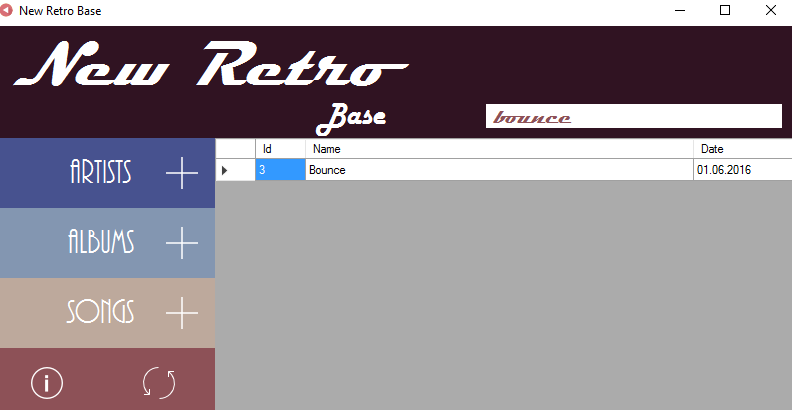
Після додавання елементів до бази, можна скористуватися пошуком, увівши запит до елементу «Search» (Рисунок 3.5) 

Рисунок 3.5

ВИСНОВКИ

При виконанні курсового проекту мною були пройдені всі основні шляхи розробки програми, від постановки задачі до налагодження програмного продукту.

У результаті розробки була створена прикладна програма «Довідник меломана» («New Retro Base») на платформі .Net мовою C#.

Програма виконує наступні функції: додавання та збереження у файл пісень, альбомів, виконавців, зчитування з файлу та пошук елементів.

Мною були досліджені такі аспекти .Net, як основні принципи ООП, використання Windows Forms, бінарна серіалізація та десеріалізація даних, опрацьовувачі подій. Мною був отриманий досвід у проектуванні, розробці на документуванні програмного продукту.

ПЕРЕЛІК ПОСИЛАНЬ

1. А. Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд. Язык программирования C#. Классика Computers Science. 4-е издание= C# Programming Language (Covering C# 4.0), 4th Ed. – СПб.: «Питер», 2012. – 784 с.
2. Бондарев В. М. «Объектно-ориентированное программирование на С#» [Текст]: учеб. пособ / В. М. Бондарев. – Х.: Смит, 2009. – 224 с.;
3. <http://msdn.com>
4. <http://flaticon.com>

Додаток А – Код програми

using System;

using System.Windows.Forms;

using MuBoxic.View;

namespace MuBoxic

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainWindow());

}

}

}

using System;

using System.Collections.Generic;

namespace MuBoxic

{

[Serializable()]

public class Album

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime Date { get; set; }

public SongList Songs;

public Album (string name, DateTime date, SongList songs)

{

Name = name;

Date = date;

Songs = songs;

}

public override string ToString()

{

return Name;

}

}

[Serializable()]

public class AlbumList : List<Album>

{

public int IdCounter;

public new void Add(Album album)

{

base.Add(album);

album.Id = IdCounter;

IdCounter++;

}

public void AddToSecondary(Album album)

{

base.Add(album);

}

}

}

using System;

using System.Collections.Generic;

namespace MuBoxic

{

[Serializable()]

public class Artist

{

public int Id { get; set; }

public string Name { get; set; }

public string Country { get; set; }

public DateTime Date { get; set; }

public AlbumList Albums;

public SongList Songs;

public Artist()

{

}

public Artist(string name, string country, DateTime date, SongList songs, AlbumList albums)

{

Name = name;

Country = country;

Date = date;

Songs = songs;

Albums = albums;

}

public override string ToString()

{

return Name;

}

}

[Serializable()]

public class ArtistList : List<Artist>

{

public int IdCounter;

public new void Add(Artist artist)

{

base.Add(artist);

artist.Id = IdCounter;

IdCounter++;

}

public void AddToSecondary(Artist artist)

{

base.Add(artist);

}

}

}

using System;

using System.Collections.Generic;

namespace MuBoxic

{

[Serializable()]

public class Song

{

public string Name { get; set; }

public DateTime Date { get; set; }

public int Id { get; set; }

public Song(string name, DateTime year)

{

Name = name;

Date = year;

}

public override string ToString()

{

return Name;

}

}

[Serializable()]

public class SongList : List<Song>

{

public int IdCounter;

public new void Add(Song song)

{

base.Add(song);

song.Id = IdCounter;

IdCounter++;

}

public void AddToSecondary(Song song)

{

base.Add(song);

}

}

}

using System;

using System.Drawing;

using System.Globalization;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Windows.Forms;

namespace MuBoxic.View

{

public partial class MainWindow : Form

{

public MainWindow()

{

InitializeComponent();

}

SongList \_songCacheList = new SongList();

AlbumList \_albumCacheList = new AlbumList();

ArtistList \_artistCacheList = new ArtistList();

const string SongBase = @"SongBase.bin";

const string AlbumBase = @"AlbumBase.bin";

const string ArtistBase = @"ArtistBase.bin";

private void addSong\_Click(object sender, EventArgs e)

{

AddSong song = new AddSong();

song.ShowDialog();

}

private void artists\_Click(object sender, EventArgs e)

{

if (File.Exists(ArtistBase))

{

Stream artistFromFile = File.Open(ArtistBase, FileMode.Open);

BinaryFormatter artistDeserializer = new BinaryFormatter();

\_artistCacheList = (ArtistList) artistDeserializer.Deserialize(artistFromFile);

artistFromFile.Close();

if (\_artistCacheList.Count != 0)

{

songView.Hide();

albumView.Hide();

artistView.Show();

artistView.DataSource = \_artistCacheList;

}

else

{

MessageBox.Show(@"There are no artists. Add some!");

}

}

else

{

MessageBox.Show(@"There are no artists. Add some!");

}

}

private void albums\_Click(object sender, EventArgs e)

{

if (File.Exists(AlbumBase))

{

Stream albumFroFile = File.Open(AlbumBase, FileMode.Open);

BinaryFormatter albumDeserializer = new BinaryFormatter();

\_albumCacheList = (AlbumList) albumDeserializer.Deserialize(albumFroFile);

albumFroFile.Close();

if (\_albumCacheList.Count != 0)

{

songView.Hide();

artistView.Hide();

albumView.Show();

albumView.DataSource = \_albumCacheList;

}

else

{

MessageBox.Show(@"There are no albums. Add some!");

}

}

else

{

MessageBox.Show(@"There are no albums. Add some!");

}

}

private void songs\_Click(object sender, EventArgs e)

{

songView.BackgroundImage = Image.FromFile(@"..\..\images\mainbg1.png");

if (File.Exists(SongBase))

{

Stream fromFile = File.Open(SongBase, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_songCacheList = (SongList) deserializer.Deserialize(fromFile);

fromFile.Close();

if (\_songCacheList.Count != 0)

{

albumView.Hide();

artistView.Hide();

songView.Show();

songView.DataSource = \_songCacheList;

}

else

{

MessageBox.Show(@"There are no songs. Add some!");

songView.Hide();

}

}

else

{

MessageBox.Show(@"There are no songs. Add some!");

}

}

private void settings\_Click(object sender, EventArgs e)

{

InfoForm info = new InfoForm();

info.Show();

}

private void addAlbum\_Click(object sender, EventArgs e)

{

AddAlbum album = new AddAlbum();

album.ShowDialog();

}

private void albumView\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

Album cache = (Album) albumView.SelectedCells[0].OwningRow.DataBoundItem;

AlbumInfo info = new AlbumInfo(cache);

info.ShowDialog();

}

private void songView\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

Song cache = (Song) songView.SelectedCells[0].OwningRow.DataBoundItem;

SongInfo info = new SongInfo(cache);

info.ShowDialog();

}

private void deleteToolStripMenuItem\_Click(object sender, EventArgs e)

{

Song cache = (Song) songView.SelectedCells[0].OwningRow.DataBoundItem;

Stream toDelete = File.Open(SongBase, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_songCacheList = (SongList) deserializer.Deserialize(toDelete);

toDelete.Close();

foreach (Song song in \_songCacheList)

{

if (song.Id == cache.Id)

{

\_songCacheList.Remove(song);

break;

}

}

FileStream toFile = File.Create(SongBase);

BinaryFormatter serializer = new BinaryFormatter();

serializer.Serialize(toFile, \_songCacheList);

toFile.Close();

if (\_songCacheList.Count != 0)

{

songView.Refresh();

songView.DataSource = \_songCacheList;

}

else

{

songView.Hide();

}

MessageBox.Show(@"Sucessfully deleted!");

}

private void showInfoOrEditToolStripMenuItem\_Click(object sender, MouseEventArgs e)

{

Song cache = (Song) songView.SelectedCells[0].OwningRow.DataBoundItem;

SongInfo info = new SongInfo(cache);

info.ShowDialog();

}

private void cellContextMenu\_Opening(object sender, System.ComponentModel.CancelEventArgs e)

{

var selected = songView.PointToClient(Cursor.Position);

DataGridView.HitTestInfo selectedCell = songView.HitTest(selected.X, selected.Y);

if (selectedCell.Type == DataGridViewHitTestType.Cell)

{

songView.ClearSelection();

songView[selectedCell.ColumnIndex, selectedCell.RowIndex].Selected = true;

}

else

{

e.Cancel = true;

}

}

private void refresh\_Click(object sender, EventArgs e)

{

if (File.Exists(SongBase))

{

Stream songsStream = File.Open(SongBase, FileMode.Open);

BinaryFormatter deserializerSongs = new BinaryFormatter();

\_songCacheList = (SongList) deserializerSongs.Deserialize(songsStream);

songsStream.Close();

songView.DataSource = \_songCacheList;

}

if (File.Exists(AlbumBase))

{

Stream albumsStream = File.Open(AlbumBase, FileMode.Open);

BinaryFormatter deserializerAlbums = new BinaryFormatter();

\_albumCacheList = (AlbumList) deserializerAlbums.Deserialize(albumsStream);

albumsStream.Close();

albumView.DataSource = \_albumCacheList;

}

if (File.Exists(ArtistBase))

{

Stream artistStream = File.Open(ArtistBase, FileMode.Open);

BinaryFormatter deserializerArtists = new BinaryFormatter();

\_artistCacheList = (ArtistList) deserializerArtists.Deserialize(artistStream);

artistStream.Close();

artistView.DataSource = \_artistCacheList;

}

}

private void SearchItem(object sender, EventArgs e)

{

string query = searchBox.Text.ToLower();

if (query == "")

{

songView.DataSource = \_songCacheList;

albumView.DataSource = \_albumCacheList;

artistView.DataSource = \_artistCacheList;

}

else if (songView.Visible)

{

SongList searchResult = new SongList();

foreach (Song song in \_songCacheList)

{

if (song.Name.ToLower().IndexOf(query, StringComparison.Ordinal) != -1

|| Convert.ToString(song.Id).IndexOf(query, StringComparison.Ordinal) != -1

|| Convert.ToString(song.Date, CultureInfo.CurrentCulture)

.IndexOf(query, StringComparison.Ordinal) != -1)

searchResult.AddToSecondary(song);

}

songView.DataSource = searchResult;

}

else if (albumView.Visible)

{

AlbumList searchResult = new AlbumList();

foreach (Album album in \_albumCacheList)

{

if (album.Name.ToLower().IndexOf(query, StringComparison.Ordinal) != -1

|| Convert.ToString(album.Id).IndexOf(query, StringComparison.Ordinal) != -1

|| Convert.ToString(album.Date, CultureInfo.CurrentCulture)

.IndexOf(query, StringComparison.Ordinal) != -1)

searchResult.AddToSecondary(album);

}

albumView.DataSource = searchResult;

}

else if (artistView.Visible)

{

ArtistList searchResult = new ArtistList();

foreach (Artist artist in \_artistCacheList)

{

if (artist.Name.ToLower().IndexOf(query, StringComparison.Ordinal) != -1

|| artist.Country.ToLower().IndexOf(query, StringComparison.Ordinal) != -1

|| Convert.ToString(artist.Id).IndexOf(query, StringComparison.Ordinal) != -1

|| Convert.ToString(artist.Date, CultureInfo.CurrentCulture)

.IndexOf(query, StringComparison.Ordinal) != -1)

searchResult.AddToSecondary(artist);

}

artistView.DataSource = searchResult;

}

}

private void albumView\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

private void searchBox\_Click(object sender, EventArgs e)

{

searchBox.Text = "";

}

private void Title1\_Click(object sender, EventArgs e)

{

searchBox.Text = @"Search...";

}

private void showInfoOrEditToolStripMenuItem1\_Click(object sender, EventArgs e)

{

Album cache = (Album)albumView.SelectedCells[0].OwningRow.DataBoundItem;

AlbumInfo info = new AlbumInfo(cache);

info.ShowDialog();

}

private void cellContextMenuAlbum\_Opening(object sender, System.ComponentModel.CancelEventArgs e)

{

var selected = albumView.PointToClient(Cursor.Position);

DataGridView.HitTestInfo selectedCell = albumView.HitTest(selected.X, selected.Y);

if (selectedCell.Type == DataGridViewHitTestType.Cell)

{

albumView.ClearSelection();

albumView[selectedCell.ColumnIndex, selectedCell.RowIndex].Selected = true;

}

else

{

e.Cancel = true;

}

}

private void addArtist\_Click(object sender, EventArgs e)

{

AddArtist artist = new AddArtist();

artist.ShowDialog();

}

private void deleteToolStripMenuItem1\_Click(object sender, EventArgs e)

{

Album cache = (Album)albumView.SelectedCells[0].OwningRow.DataBoundItem;

Stream toDelete = File.Open(AlbumBase, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_albumCacheList = (AlbumList)deserializer.Deserialize(toDelete);

toDelete.Close();

foreach (Album album in \_albumCacheList)

{

if (album.Id == cache.Id)

{

\_albumCacheList.Remove(album);

break;

}

}

FileStream toFile = File.Create(AlbumBase);

BinaryFormatter serializer = new BinaryFormatter();

serializer.Serialize(toFile, \_albumCacheList);

toFile.Close();

if (\_albumCacheList.Count != 0)

{

albumView.Refresh();

albumView.DataSource = \_albumCacheList;

}

else

{

albumView.Hide();

}

MessageBox.Show(@"Sucessfully deleted!");

}

private void artistView\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

Artist cache = (Artist)artistView.SelectedCells[0].OwningRow.DataBoundItem;

ArtistInfo info = new ArtistInfo(cache);

info.ShowDialog();

}

private void cellContextMenuArtist\_Opening(object sender, System.ComponentModel.CancelEventArgs e)

{

var selected = artistView.PointToClient(Cursor.Position);

DataGridView.HitTestInfo selectedCell = artistView.HitTest(selected.X, selected.Y);

if (selectedCell.Type == DataGridViewHitTestType.Cell)

{

artistView.ClearSelection();

artistView[selectedCell.ColumnIndex, selectedCell.RowIndex].Selected = true;

}

else

{

e.Cancel = true;

}

}

private void showInfoOrEditToolStripMenuItem2\_Click(object sender, EventArgs e)

{

Artist cache = (Artist)artistView.SelectedCells[0].OwningRow.DataBoundItem;

ArtistInfo info = new ArtistInfo(cache);

info.ShowDialog();

}

private void deleteToolStripMenuItem2\_Click(object sender, EventArgs e)

{

Artist cache = (Artist)artistView.SelectedCells[0].OwningRow.DataBoundItem;

Stream toDelete = File.Open(ArtistBase, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_artistCacheList = (ArtistList)deserializer.Deserialize(toDelete);

toDelete.Close();

foreach (Artist artist in \_artistCacheList)

{

if (artist.Id == cache.Id)

{

\_artistCacheList.Remove(artist);

break;

}

}

FileStream toFile = File.Create(ArtistBase);

BinaryFormatter serializer = new BinaryFormatter();

serializer.Serialize(toFile, \_artistCacheList);

toFile.Close();

if (\_artistCacheList.Count != 0)

{

artistView.Refresh();

artistView.DataSource = \_artistCacheList;

}

else

{

artistView.Hide();

}

MessageBox.Show(@"Sucessfully deleted!");

}

}

}

using System;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Windows.Forms;

namespace MuBoxic.View

{

public partial class AddSong : Form

{

const string FileName = @"SongBase.bin";

public AddSong()

{

InitializeComponent();

}

private SongList \_cacheList = new SongList();

private void submit\_Click(object sender, EventArgs e)

{

if (File.Exists(FileName))

{

Stream fromFile = File.Open(FileName, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_cacheList = (SongList)deserializer.Deserialize(fromFile);

fromFile.Close();

}

else

{

Stream create = File.Create(FileName);

create.Close();

}

if (name.Text.Length != 0)

{

int spaces = 0;

for (int i = 0; i < name.TextLength; i++)

{

if (name.Text[i] == ' ')

{

spaces++;

}

}

if (spaces != name.TextLength)

{

Song cacheSong = new Song(name.Text, date.Value.Date);

\_cacheList.Add(cacheSong);

MessageBox.Show(@"Done!");

name.Text = "";

}

else

{

MessageBox.Show(@"Enter a correct name");

}

}

else

{

MessageBox.Show(@"Enter the name of the song");

}

if (File.Exists(FileName))

{

FileStream toFile = File.OpenWrite(FileName);

BinaryFormatter serializer = new BinaryFormatter();

serializer.Serialize(toFile, \_cacheList);

toFile.Close();

}

else

{

FileStream toFile = File.Create(FileName);

BinaryFormatter serializer = new BinaryFormatter();

serializer.Serialize(toFile, \_cacheList);

toFile.Close();

}

}

}

}

using System;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Windows.Forms;

namespace MuBoxic.View

{

public partial class AddAlbum : Form

{

const string SongBase = @"SongBase.bin";

const string AlbumBase = @"AlbumBase.bin";

public AddAlbum()

{

InitializeComponent();

}

private SongList \_songCacheList = new SongList();

private AlbumList \_albumCacheList = new AlbumList();

private void AddAlbum\_Load(object sender, EventArgs e)

{

if (File.Exists(SongBase))

{

Stream fromFile = File.Open(SongBase, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_songCacheList = (SongList)deserializer.Deserialize(fromFile);

fromFile.Close();

songs.DataSource = \_songCacheList;

}

else

{

MessageBox.Show(@"There are no songs. You can not add an album!");

Close();

}

}

private void submit\_Click(object sender, EventArgs e)

{

\_songCacheList.Clear();

foreach (Song song in songs.CheckedItems)

{

\_songCacheList.AddToSecondary(song);

}

if (File.Exists(AlbumBase))

{

Stream fromFile = File.Open(AlbumBase, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_albumCacheList = (AlbumList)deserializer.Deserialize(fromFile);

fromFile.Close();

}

else

{

Stream create = File.Create(AlbumBase);

create.Close();

}

if (name.TextLength != 0)

{

int spaces = 0;

for (int i = 0; i < name.TextLength; i++)

{

if (name.Text[i] == ' ') spaces++;

}

if (spaces != name.TextLength)

{

Album cacheAlbum = new Album(name.Text, date.Value.Date, \_songCacheList);

\_albumCacheList.Add(cacheAlbum);

MessageBox.Show(@"Done!");

name.Text = "";

}

else

{

MessageBox.Show(@"Enter a correct name");

}

}

else

{

MessageBox.Show(@"Enter the name of the album");

}

if (File.Exists(AlbumBase))

{

FileStream toFile = File.OpenWrite(AlbumBase);

BinaryFormatter serializer = new BinaryFormatter();

serializer.Serialize(toFile, \_albumCacheList);

toFile.Close();

}

else

{

FileStream toFile = File.Create(AlbumBase);

BinaryFormatter serializer = new BinaryFormatter();

serializer.Serialize(toFile, \_albumCacheList);

toFile.Close();

}

\_songCacheList.Clear();

}

}

}

using System;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Windows.Forms;

namespace MuBoxic.View

{

public partial class AddArtist : Form

{

const string SongBase = @"SongBase.bin";

const string AlbumBase = @"AlbumBase.bin";

const string ArtistBase = @"ArtistBase.bin";

public AddArtist()

{

InitializeComponent();

}

private SongList \_songCacheList = new SongList();

private AlbumList \_albumCacheList = new AlbumList();

private ArtistList \_artistCacheList = new ArtistList();

private void AddArtist\_Load(object sender, EventArgs e)

{

if (File.Exists(SongBase))

{

Stream songFromFile = File.Open(SongBase, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_songCacheList = (SongList) deserializer.Deserialize(songFromFile);

songFromFile.Close();

songs.DataSource = \_songCacheList;

if (File.Exists(AlbumBase))

{

Stream albumFromFile = File.Open(AlbumBase, FileMode.Open);

\_albumCacheList = (AlbumList) deserializer.Deserialize(albumFromFile);

albumFromFile.Close();

albums.DataSource = \_albumCacheList;

}

}

else

{

MessageBox.Show(@"There are no songs and albums. You can not add an artist!");

Close();

}

}

private void submit\_Click(object sender, EventArgs e)

{

\_songCacheList.Clear();

\_albumCacheList.Clear();

foreach (Song song in songs.CheckedItems)

{

\_songCacheList.AddToSecondary(song);

}

foreach (Album album in albums.CheckedItems)

{

\_albumCacheList.AddToSecondary(album);

}

if (File.Exists(ArtistBase))

{

Stream fromFile = File.Open(ArtistBase, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_artistCacheList = (ArtistList) deserializer.Deserialize(fromFile);

fromFile.Close();

}

else

{

Stream create = File.Create(ArtistBase);

create.Close();

}

if (name.TextLength != 0 && country.TextLength != 0)

{

int spacesInName = 0;

int spacesInCountry = 0;

for (int i = 0; i < name.TextLength; i++)

{

if (name.Text[i] == ' ') spacesInName++;

}

for (int i = 0; i < country.TextLength; i++)

{

if (country.Text[i] == ' ') spacesInCountry++;

}

if (spacesInName != name.TextLength && spacesInCountry != country.TextLength)

{

Artist cacheArtist = new Artist();

if (\_songCacheList.Count != 0 && \_albumCacheList.Count != 0)

{

cacheArtist.Name = name.Text;

cacheArtist.Country = country.Text;

cacheArtist.Date = date.Value.Date;

cacheArtist.Songs = \_songCacheList;

cacheArtist.Albums = \_albumCacheList;

}

else if (\_songCacheList.Count != 0)

{

cacheArtist.Name = name.Text;

cacheArtist.Country = country.Text;

cacheArtist.Date = date.Value.Date;

cacheArtist.Songs = \_songCacheList;

cacheArtist.Albums = new AlbumList();

}

else if (\_albumCacheList.Count != 0)

{

cacheArtist.Name = name.Text;

cacheArtist.Country = country.Text;

cacheArtist.Date = date.Value.Date;

cacheArtist.Albums = \_albumCacheList;

cacheArtist.Songs = new SongList();

}

\_artistCacheList.Add(cacheArtist);

MessageBox.Show(@"Done!");

name.Text = "";

country.Text = "";

}

else

{

MessageBox.Show(@"Enter a correct name and country");

}

}

else

{

MessageBox.Show(@"Enter the name of the artist and the country");

}

if (File.Exists(ArtistBase))

{

FileStream artistToFile = File.OpenWrite(ArtistBase);

BinaryFormatter serializer = new BinaryFormatter();

serializer.Serialize(artistToFile, \_artistCacheList);

artistToFile.Close();

}

else

{

FileStream artistToFile = File.Create(ArtistBase);

BinaryFormatter serializer = new BinaryFormatter();

serializer.Serialize(artistToFile, \_artistCacheList);

artistToFile.Close();

}

\_songCacheList.Clear();

\_albumCacheList.Clear();

}

}

}

using System;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Windows.Forms;

namespace MuBoxic.View

{

public partial class SongInfo : Form

{

const string SongBase = @"SongBase.bin";

const string AlbumBase = @"AlbumBase.Bin";

private string \_cacheName;

private DateTime \_cacheDate;

private SongList \_toEditSongList = new SongList();

private readonly Song \_toShowSong;

private AlbumList \_cacheAlbumList = new AlbumList();

private readonly AlbumList \_toShowAlbumList = new AlbumList();

public SongInfo(Song cache)

{

\_toShowSong = cache;

InitializeComponent();

}

private void SongInfo\_Load(object sender, EventArgs e)

{

name.Text = \_toShowSong.Name;

id.Text = @"Id: " + \_toShowSong.Id;

date.Value = \_toShowSong.Date;

if (File.Exists(AlbumBase))

{

Stream showAlbum = File.Open(AlbumBase, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_cacheAlbumList = (AlbumList) deserializer.Deserialize(showAlbum);

showAlbum.Close();

foreach (Album album in \_cacheAlbumList)

{

foreach (Song song in album.Songs)

{

if (song.Id == \_toShowSong.Id) \_toShowAlbumList.AddToSecondary(album);

}

}

if (\_toShowAlbumList.Count != 0)

{

fromAlbumsList.DataSource = \_toShowAlbumList;

}

else

{

fromAlbumsList.Hide();

fromAlbum.Text = "There are no albums\nthat contain this song";

}

}

else

{

fromAlbumsList.Hide();

fromAlbum.Text = @"There are no albums";

}

}

private void edit\_Click(object sender, EventArgs e)

{

edit.Enabled = false;

date.Enabled = true;

name.Enabled = true;

submit.Enabled = true;

\_cacheName = name.Text;

\_cacheDate = date.Value;

}

private void submit\_Click(object sender, EventArgs e)

{

if ((name.Text != \_cacheName || date.Value != \_cacheDate) && name.TextLength != 0)

{

int spaces = 0;

for (int i = 0; i < name.TextLength; i++)

{

if (name.Text[i] == ' ') spaces ++;

}

if (spaces == name.TextLength)

{

MessageBox.Show(@"Enter a correct name");

}

else

{

Stream toEdit = File.Open(SongBase, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_toEditSongList = (SongList) deserializer.Deserialize(toEdit);

toEdit.Close();

foreach (Song song in \_toEditSongList)

{

if (\_toShowSong.Id == song.Id)

{

song.Name = name.Text;

song.Date = date.Value;

}

}

FileStream toFile = File.OpenWrite(SongBase);

BinaryFormatter serializer = new BinaryFormatter();

serializer.Serialize(toFile, \_toEditSongList);

toFile.Close();

}

MessageBox.Show(@"Sucessfully edited!");

Close();

}

else

{

MessageBox.Show(@"You did not commit any changes");

}

}

}

}

using System;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Windows.Forms;

namespace MuBoxic.View

{

public partial class AlbumInfo : Form

{

const string AlbumBase = @"AlbumBase.bin";

const string SongBase = @"SongBase.bin";

private DateTime \_cacheDate;

private string \_cacheName;

private SongList \_toEditSongList;

private AlbumList \_toEditAlbumList = new AlbumList();

private readonly Album \_toShowAlbum;

public AlbumInfo(Album cache)

{

\_toShowAlbum = cache;

InitializeComponent();

}

private void AlbumInfo\_Load(object sender, EventArgs e)

{

name.Text = \_toShowAlbum.Name;

date.Value = \_toShowAlbum.Date;

id.Text = @"Id: " + \_toShowAlbum.Id;

songs.DataSource = \_toShowAlbum.Songs;

}

private void edit\_Click(object sender, EventArgs e)

{

edit.Enabled = false;

date.Enabled = true;

name.Enabled = true;

submit.Enabled = true;

songs.Visible = false;

editSongs.Visible = true;

Stream toCheckList = File.Open(SongBase, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_toEditSongList = (SongList)deserializer.Deserialize(toCheckList);

toCheckList.Close();

\_cacheDate = date.Value;

\_cacheName = name.Text;

editSongs.DataSource = \_toEditSongList;

}

private void submit\_Click(object sender, EventArgs e)

{

if ((name.Text != \_cacheName || date.Value != \_cacheDate || songs.DataSource != editSongs.CheckedItems) &&

name.TextLength != 0)

{

int spaces = 0;

for (int i = 0; i < name.TextLength; i++)

{

if (name.Text[i] == ' ') spaces++;

}

if (spaces == name.TextLength)

{

MessageBox.Show(@"Enter a correct name");

}

else

{

Stream toEdit = File.Open(AlbumBase, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_toEditAlbumList = (AlbumList) deserializer.Deserialize(toEdit);

toEdit.Close();

foreach (Album album in \_toEditAlbumList)

{

if (\_toShowAlbum.Id == album.Id)

{

album.Name = name.Text;

album.Date = date.Value;

if (editSongs.CheckedItems.Count != 0)

{

album.Songs.Clear();

foreach (Song song in editSongs.CheckedItems)

{

album.Songs.AddToSecondary(song);

}

}

}

}

FileStream toFile = File.OpenWrite(AlbumBase);

BinaryFormatter serializer = new BinaryFormatter();

serializer.Serialize(toFile, \_toEditAlbumList);

toFile.Close();

}

MessageBox.Show(@"Successfully edited!");

Close();

}

else

{

MessageBox.Show(@"You did not commit any changes");

}

}

private void showInfoOrEditToolStripMenuItem\_Click(object sender, EventArgs e)

{

Song cache = (Song)songs.SelectedCells[0].OwningRow.DataBoundItem;

SongInfo info = new SongInfo(cache);

info.ShowDialog();

}

private void cellContextMenuAlbumInfo\_Opening(object sender, System.ComponentModel.CancelEventArgs e)

{

var selected = songs.PointToClient(Cursor.Position);

DataGridView.HitTestInfo selectedCell = songs.HitTest(selected.X, selected.Y);

if (selectedCell.Type == DataGridViewHitTestType.Cell)

{

songs.ClearSelection();

songs[selectedCell.ColumnIndex, selectedCell.RowIndex].Selected = true;

}

else

{

e.Cancel = true;

}

}

}

}

using System;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Windows.Forms;

namespace MuBoxic.View

{

public partial class ArtistInfo : Form

{

const string ArtistBase = @"ArtistBase.bin";

const string AlbumBase = @"AlbumBase.bin";

const string SongBase = @"SongBase.bin";

private string \_cacheName;

private DateTime \_cacheDate;

private string \_cacheCountry;

private SongList \_toEditSongList;

private AlbumList \_toEditAlbumList;

private SongList \_allSongsList = new SongList();

private ArtistList \_toeditArtistList = new ArtistList();

private readonly Artist \_toShowArtist;

public ArtistInfo(Artist cache)

{

\_toShowArtist = cache;

InitializeComponent();

}

private void ArtistInfo\_Load(object sender, EventArgs e)

{

name.Text = \_toShowArtist.Name;

country.Text = \_toShowArtist.Country;

id.Text = @"Id: " + \_toShowArtist.Id;

if (\_toShowArtist.Albums != null)

{

foreach (Album album in \_toShowArtist.Albums)

{

foreach (Song song in album.Songs)

{

\_allSongsList.AddToSecondary(song);

}

}

}

if (\_toShowArtist.Songs != null)

{

foreach (Song song in \_toShowArtist.Songs)

{

\_allSongsList.AddToSecondary(song);

}

}

songView.DataSource = \_allSongsList;

albumView.DataSource = \_toShowArtist.Albums;

}

private void edit\_Click(object sender, EventArgs e)

{

edit.Enabled = false;

date.Enabled = true;

name.Enabled = true;

country.Enabled = true;

submit.Enabled = true;

songView.Visible = false;

albumView.Visible = false;

editSongs.Visible = true;

editAlbums.Visible = true;

Stream toSongCheckList = File.Open(SongBase, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_toEditSongList = (SongList) deserializer.Deserialize(toSongCheckList);

toSongCheckList.Close();

if (File.Exists(AlbumBase))

{

Stream toAlbumCheckList = File.Open(AlbumBase, FileMode.Open);

\_toEditAlbumList = (AlbumList) deserializer.Deserialize(toAlbumCheckList);

toAlbumCheckList.Close();

}

\_cacheDate = date.Value;

\_cacheName = name.Text;

\_cacheCountry = country.Text;

editSongs.DataSource = \_toEditSongList;

editAlbums.DataSource = \_toEditAlbumList;

}

private void submit\_Click(object sender, EventArgs e)

{

if ((name.Text != \_cacheName

|| date.Value != \_cacheDate

|| country.Text != \_cacheCountry

|| songView.DataSource != editSongs.CheckedItems

|| albumView.DataSource != editAlbums.CheckedItems)

&& name.TextLength != 0)

{

int spacesInName = 0;

int spacesInCountry = 0;

for (int i = 0; i < name.TextLength; i++)

{

if (name.Text[i] == ' ') spacesInName++;

}

for (int i = 0; i < country.TextLength; i++)

{

if (country.Text[i] == ' ') spacesInCountry++;

}

if (spacesInCountry == country.TextLength || spacesInName == name.TextLength)

{

MessageBox.Show(@"Enter a correct name and country");

}

else

{

Stream toEdit = File.Open(ArtistBase, FileMode.Open);

BinaryFormatter deserializer = new BinaryFormatter();

\_toeditArtistList = (ArtistList) deserializer.Deserialize(toEdit);

toEdit.Close();

foreach (Artist artist in \_toeditArtistList)

{

if (\_toShowArtist.Id == artist.Id)

{

artist.Name = name.Text;

artist.Date = date.Value;

artist.Country = country.Text;

if (editSongs.CheckedItems.Count != 0)

{

if (artist.Songs != null)

{

artist.Songs.Clear();

foreach (Song song in editSongs.CheckedItems)

{

artist.Songs.AddToSecondary(song);

}

}

}

if (editAlbums.CheckedItems.Count != 0)

{

artist.Albums.Clear();

foreach (Album album in editAlbums.CheckedItems)

{

artist.Albums.AddToSecondary(album);

}

}

}

}

FileStream toFile = File.OpenWrite(ArtistBase);

BinaryFormatter serializer = new BinaryFormatter();

serializer.Serialize(toFile, \_toeditArtistList);

toFile.Close();

}

MessageBox.Show(@"Successfully edited!");

Close();

}

else

{

MessageBox.Show(@"You did not commit any changes");

}

}

private void showInfoToolStripMenuItem1\_Click(object sender, EventArgs e)

{

Song cache = (Song)songView.SelectedCells[0].OwningRow.DataBoundItem;

SongInfo info = new SongInfo(cache);

info.ShowDialog();

}

private void cellContextMenuSongInfo\_Opening(object sender, System.ComponentModel.CancelEventArgs e)

{

var selected = songView.PointToClient(Cursor.Position);

DataGridView.HitTestInfo selectedCell = songView.HitTest(selected.X, selected.Y);

if (selectedCell.Type == DataGridViewHitTestType.Cell)

{

songView.ClearSelection();

songView[selectedCell.ColumnIndex, selectedCell.RowIndex].Selected = true;

}

else

{

e.Cancel = true;

}

}

private void cellContextMenuAlbumInfo\_Opening(object sender, System.ComponentModel.CancelEventArgs e)

{

var selected = albumView.PointToClient(Cursor.Position);

DataGridView.HitTestInfo selectedCell = albumView.HitTest(selected.X, selected.Y);

if (selectedCell.Type == DataGridViewHitTestType.Cell)

{

albumView.ClearSelection();

albumView[selectedCell.ColumnIndex, selectedCell.RowIndex].Selected = true;

}

else

{

e.Cancel = true;

}

}

private void showInfoToolStripMenuItem\_Click(object sender, EventArgs e)

{

Album cache = (Album)albumView.SelectedCells[0].OwningRow.DataBoundItem;

AlbumInfo info = new AlbumInfo(cache);

info.ShowDialog();

}

private void albumView\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

Album cache = (Album)albumView.SelectedCells[0].OwningRow.DataBoundItem;

AlbumInfo info = new AlbumInfo(cache);

info.ShowDialog();

}

private void songView\_CellDoubleClick(object sender, DataGridViewCellEventArgs e)

{

Song cache = (Song)songView.SelectedCells[0].OwningRow.DataBoundItem;

SongInfo info = new SongInfo(cache);

info.ShowDialog();

}

}

}

using System.Windows.Forms;

namespace MuBoxic.View

{

public partial class InfoForm : Form

{

public InfoForm()

{

InitializeComponent();

}

}

}