### Modulul 6. Supraîncărcarea operatorilor în C#.

#### Introducere în supraîncărcarea operatorilor

Supraîncărcarea operatorilor are în esență exact aceleași funcții și utilitate ca și supraîncărcarea metodelor pe care am studiat-o în Modulul 3.

Supraîncărcarea operatorilor se întâmplă atunci când programatorul declară explicit și intenționat alte metode de a utiliza un operator, în dependență de variabilele cu care acesta va interacționa.

Spre exemplu știm bine că operatorul + răspunde de adunarea tipurilor și concatinarea stringurilor și caracterelor, dar puteți să creați pentru acesta o metodă supraîncărcată care vă va permite să folosiți operatorul + pentru a aduna două structuri sau clase de exemplu.

Operatorii care pot fi supraîncărcați în C#:

[+](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#unary-plus-and-minus-operators), [-](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#unary-plus-and-minus-operators), [!](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#logical-negation-operator-), [~](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#bitwise-complement-operator-), [++](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#increment-operator-), [--](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#decrement-operator---), [true](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/true-false-operators), [false](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/true-false-operators), [\*](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#multiplication-operator-), [/](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#division-operator-), [%](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#remainder-operator-), [&](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#logical-and-operator-), [|](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#logical-or-operator-), [^](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#logical-exclusive-or-operator-), [<<](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#left-shift-operator-), [>>](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#right-shift-operator-), [==](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/equality-operators#equality-operator-), [!=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/equality-operators#inequality-operator-), [<](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/comparison-operators#less-than-operator-), [>](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/comparison-operators#greater-than-operator-), [<=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/comparison-operators#less-than-or-equal-operator-), [>=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/comparison-operators#greater-than-or-equal-operator-)

Operatorii care NU pot fi supraîncărcați:

[&&](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#conditional-logical-and-operator-), [||](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#conditional-logical-or-operator-), [ ], [+=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#compound-assignment), [-=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#compound-assignment), [\*=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#compound-assignment), [/=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#compound-assignment), [%=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#compound-assignment), [&=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#compound-assignment), [|=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#compound-assignment), [^=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#compound-assignment), [<<=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#compound-assignment), [>>=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#compound-assignment), [^x](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/member-access-operators#index-from-end-operator-), [x = y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/assignment-operator), [x.y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/member-access-operators#member-access-expression-), [x?.y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/member-access-operators#null-conditional-operators--and-), [c ? t : f](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/conditional-operator), [x ?? y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-coalescing-operator), [x ??= y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-coalescing-operator), [x..y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/member-access-operators#range-operator-), [x->y](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/pointer-related-operators#pointer-member-access-operator--), [=>](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/lambda-operator), [f(x)](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/member-access-operators#invocation-expression-), [as](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#as-operator), [await](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/await), [checked](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/checked), [unchecked](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/unchecked), [default](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/default), [delegate](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/delegate-operator), [is](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#is-operator), [nameof](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/nameof), [new](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/new-operator), [sizeof](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/sizeof), [stackalloc](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/stackalloc), [typeof](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#typeof-operator)

Sintaxa declarării:

1)Nivelul de acces 2)Tipul de date 3) cuv cheie **operator** 4) Simbolul Operatorului 5) (parametri)

{

//Definirea logicii

}

*Exemplu practic:*

using System;

namespace Calculator

{

class Calculator

{

public int number1, number2;

public Calculator(int num1, int num2)

{

number1 = num1;

number2 = num2;

}

public static Calculator operator -(Calculator \_calc1, Calculator \_calc2)

{

\_calc1.number1 -= \_calc2.number1;

\_calc1.number2 -= \_calc2.number2;

return \_calc1;

}

public static Calculator operator +(Calculator \_calc1, Calculator \_calc2)

{

\_calc1.number1 += \_calc2.number1;

\_calc1.number2 += \_calc2.number2;

return \_calc1;

}

public void Print()

{

Console.WriteLine("Number1 = " + number1);

Console.WriteLine("Number2 = " + number2);

}

}

class Executie

{

static void Main()

{

Calculator calc1 = new Calculator(15, -25);

Calculator calc2 = new Calculator(5, 15);

calc1 = calc1 - calc2;

calc1.Print();

}

}

}

În programul expus am utilizat supraîncărcarea operatorului + și - pentru a crea o metodă adițională de a-i folosi, și anume de a aduna și scădea două obiecte care ambele dețin clasa Calculator.

Acest lucru ne permite să facem operația de + și - între două obiecte care în mod normal nu ar permite această operație, și precum am definit în cod când vom face una din aceste operații number1 din clasa 1 va fi adunat(sau scăzut) cu number 1 din clasa 2, și numărul 2 respectiv.

În metoda Main, am creat două obiecte de tip calculator cu parametri diferiți și am executat operația -, care a dus la ceea că fiecare număr din calc1 a fost scăzut cu numărul respectiv din calc2.,

#### Supraîncărcarea operatorilor unari

Operatorii unari care pot fi supraîncărcați în C# sunt:+,-, !, ~, ++, --. Operatorii unari primesc UN SINGUR parametru.

*Exemplu +:*

class Adunare

{

public int variabila;

public Adunare(int \_var)

{

variabila = \_var;

}

public static Adunare operator +(Adunare \_prog1)

{

\_prog1.variabila += 5;

return \_prog1;

}

static void Main()

{

Adunare obj1 = new Adunare(5);

obj1 = +obj1;

WriteLine(obj1.variabila);

Beep();

}

}

*Exemplu -:*

class Diferenta

{

public int numar;

public Diferenta(int \_input)

{

numar = \_input;

}

public static Diferenta operator -(int \_input, Diferenta \_prog1)

{

\_prog1.numar = -\_prog1.numar \* 2;

return \_prog1;

}

static void Main()

{

Diferenta prog1 = new Diferenta(5);

prog1 = 0 - prog1;

WriteLine(prog1.numar);

Beep();

}

}

*Exemplu !(negație):*

class Negatie

{

public bool valoare;

public Negatie(bool \_input)

{

valoare = \_input;

}

public static Negatie operator !(Negatie \_prog1)

{

if (\_prog1.valoare) \_prog1.valoare = !\_prog1.valoare;

return \_prog1;

}

static void Main()

{

Negatie prog1 = new Negatie(true);

Negatie prog2 = new Negatie(false);

prog1 = !prog1;

prog2 = !prog2;

WriteLine(prog1.valoare);

WriteLine(prog2.valoare);

Beep();

}

}

#### Supraîncărcarea operatorilor binari

Spre deosebire de operatorii unari, operatorii binari vor primi două variabile de intrare.

**Observație: La utilizarea operatorilor binari, trebuie să avem minim o variabilă de intrare care este de tipul clasei.**

În C# următorii operatori binari pot fi supraîncărcați: [+](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/addition-operator), [-](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/subtraction-operator), [\*](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#multiplication-operator-), [/](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#division-operator-), [%](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#remainder-operator-),[<<,](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#left-shift-operator-) >>

*Exemplu + binar:*

using static System.Console;

namespace OperatoriUnari

{

class Program

{

public int[] tablou;

public Program(int[] \_input)

{

tablou = \_input;

}

public static Program operator +(Program \_prog1, Program \_prog2)

{

for (int i = 0; i < \_prog1.tablou.Length; i++)

{

\_prog1.tablou[i] += \_prog2.tablou[i];

}

return \_prog1;

}

static void Main(string[] args)

{

Program prog1 = new Program(new int[]{ 13, 2, -5 });

Program prog2 = new Program(new int[] { 15, 6, 8 });

prog1 += prog2;

foreach (int numar in prog1.tablou)

{

WriteLine(numar);

}

Beep();

}

}

}

*Exemplu - binar:*

class Scadere

{

public int numar;

public Scadere(int \_input)

{

numar = \_input;

}

public static Scadere operator -(int \_numar, Scadere \_obj)

{

\_obj.numar = \_numar - \_obj.numar;

return \_obj;

}

static void Main()

{

Scadere prog1 = new Scadere(5);

prog1 = 10 - prog1;

WriteLine(prog1.numar);

Beep();

}

}

#### Supraîncărcarea operatorilor de comparație

[==](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/equality-operators#equality-operator-), [!=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/equality-operators#inequality-operator-), [<](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/comparison-operators#less-than-operator-), [>](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/comparison-operators#greater-than-operator-), [<=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/comparison-operators#less-than-or-equal-operator-), [>=](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/comparison-operators#greater-than-or-equal-operator-)

Supraîncărcarea operatorilor de comparație ne oferă o metodă comodă de a integra funcționalități de genul comparării clasei cu un număr, sau comparării a două clase între ele după un anumită variabilă, sau verificarea dacă un obiect de tipul clasei coincide cu alt obiect după o variabilă ID din ele etc.

*Exemplu pentru ==, !=, >, <:*

using static System.Console;

namespace OperatoriComparatie

{

class Egal

{

public int ID;

public int valoare;

//Constructor

public Egal(int[] \_input)

{

ID = \_input[0];

valoare = \_input[1];

}

//Operator supraincarcat egal si nu egal

public static bool operator ==(Egal \_prog1, Egal \_prog2)

{

return (\_prog1.ID == \_prog2.ID);

}

public static bool operator !=(Egal \_prog1, Egal \_prog2)

{

return (\_prog1.ID != \_prog2.ID);

}

//Operator supraincarcat mai mare mai mic

public static bool operator >(Egal \_prog1, Egal \_prog2)

{

return (\_prog1.valoare > \_prog2.valoare);

}

public static bool operator <(Egal \_prog1, Egal \_prog2)

{

return (\_prog1.valoare < \_prog2.valoare);

}

static void Main(string[] args)

{

Egal prog1 = new Egal(new int[] { 5, 7});

Egal prog2 = new Egal(new int[] { 5, 8});

WriteLine("Obiectul 1 are acelasi ID ca si 2:"+(prog1 == prog2));

WriteLine("Obiectul 1 are ID diferit de 2:"+(prog1 != prog2));

WriteLine("\n");

WriteLine("Obiectul 1 are valoarea mai mare ca 2:" + (prog1 > prog2));

WriteLine("Obiectul 1 are valoarea mai mica ca 2:" + (prog1 < prog2));

Beep();

}

}

}

#### Supraîncărcarea operatorilor logici ([&](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#logical-and-operator-), [|](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#logical-or-operator-) , [^](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators#logical-exclusive-or-operator-))

Ca și variantele lor simple, operatorii logici supraîncărcați vor fi utilizați pentru execuția operațiilor între valori true/false.

*Exemplu operatori logici:*

namespace OperatoriLogici

{

class Client

{

public int cheltuieli;

//Constructor

public Client(int \_input)

{

cheltuieli = \_input;

}

//Operatori logici supraincarcati

public static bool operator &(Client \_client1, Client \_client2)

{

bool activ1 = (\_client1.cheltuieli > 0);

bool activ2 = (\_client2.cheltuieli > 0);

return activ1 & activ2;

}

public static bool operator |(Client \_client1, Client \_client2)

{

bool activ1 = (\_client1.cheltuieli > 0);

bool activ2 = (\_client2.cheltuieli > 0);

return activ1 | activ2;

}

public static bool operator ^(Client \_client1, Client \_client2)

{

bool activ1 = (\_client1.cheltuieli > 0);

bool activ2 = (\_client2.cheltuieli > 0);

return activ1 ^ activ2;

}

static void Main(string[] args)

{

Client client1 = new Client(10000);

Client client2 = new Client(0);

WriteLine("Ambii clienti sunt activi:" + (client1 & client2));

WriteLine("Unul dintre clienti este activ:" + (client1 | client2));

WriteLine("Clientii au statut diferit de activitate:" + (client1 ^ client2));

Beep();

}

}

}

#### Supraîncărcarea operatorilor true/false

Ca și alți operatori C#, true și false pot fi supraîncărcați ca să returneze o anumită valoarea în anumite circumstanțe. Spre exemplu putem utiliza supraîncărcarea pentru a crea o clasă Vector și de a verifica dacă vectorul nostru este 0 sau nu printr-o simplă variabilă true/false.

*Exemplu true/false:*

class Vector

{

public int X;

public int Y;

public Vector(int \_x, int \_y)

{

X = \_x;

Y = \_y;

}

public static bool operator true(Vector v)

{

if ((v.X != 0) || (v.Y != 0))

return true;

else

return false;

}

public static bool operator false(Vector v)

{

if ((v.X == 0) && (v.Y == 0))

return true;

else

return false;

}

static void Main(string[] args)

{

Vector test = new Vector(5, 0);

if (test) WriteLine("Vectorul introdus nu este 0.");

else WriteLine("Vectorul introdus este egal cu 0.");

}

}

#### Supraîncărcarea operatorilor de conversie

După cum am discutat în modulul 2 în C# există 2 tipuri de conversie, implicită și explicită. Ambele tipuri pot fi supraîncărcate pentru a oferi mai multe posibilități de utilizare.

Sintaxa

public static implicit operator double(Vector \_vector)

{

return \_vector.lungime;

}

*Exemplu conversie implicită:*

class Vector

{

public int X;

public int Y;

public double lungime;

public Vector(int \_x, int \_y)

{

X = \_x;

Y = \_y;

lungime = Math.Sqrt((Math.Pow(X, 2) + Math.Pow(Y, 2)));

}

public static implicit operator double(Vector \_vector)

{

return \_vector.lungime;

}

static void Main()

{

Vector test = new Vector(5, 5);

double vectorLungime = test;

WriteLine("Lungimea vectorului introdus este de:{0}", vectorLungime);

}

}

*Exemplu conversie explicită:*

class Vector

{

public int X;

public int Y;

public float lungime;

public Vector(int \_x, int \_y)

{

X = \_x;

Y = \_y;

lungime = (float)Math.Sqrt((Math.Pow(X, 2) + Math.Pow(Y, 2)));

}

public static explicit operator float(Vector \_vector)

{

return (float)\_vector.lungime;

}

static void Main()

{

Vector test = new Vector(5, 5);

float vectorLungime = (float)test;

WriteLine("Lungimea vectorului introdus este de:{0}", vectorLungime.ToString("F2"));

}

}

### Probleme pentru rezolvare.

### **Program 1:**

### Scrieți un program în care veți supraîncărca **operația - (minus) binară**.

### Operatorul va executa diferența doar dacă rezultatul va fi mai mare ca 0.

### În metoda main folosiți operatorul supraîncărcat în două cazuri, în primul diferența va fi mai mare ca 0, în al doilea mai mică.

### Afișați ambele rezultate.

### **Program 2:**

### Scrieți un program în care veți supraîncărca operația **++ (incrementare) unară**.

### Operatorul va executa adunarea cu 3 doar dacă parametrul este impar.

### În metoda main creați două obiecte de tipul clasei voastre, unul cu parametru par, altul impar.

### Folosiți operatorul supraîncărcat și afișați ambele rezultate.

### **Program 3:**

### Scrieți un program în care veți supraîncărca **operația / (împărțire) binară**.

### Integrați un try catch în supraîncărcare pentru a detecta împărțirea cu 0.

### Dacă identificați împărțirea cu 0, înlocuiți 0 cu alt număr și repetați operația.

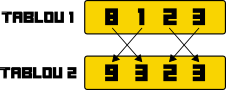
### Afișați rezultatul nou.

### **Program 4:**

### Scrieți un program în care veți supraîncărca **operația \* (produs) binară**.

### Operația va executa produsul a două tablouri numerice în formă încrucișată.

(vedeți schema mai jos - puteți folosi orice valori).



### Afișați tabloul rezultat.

### **Program 5:**

### Creați o clasă în care veți avea doi parametri: număr de ordine și valoare

### Supraîncărcați **operația == (egalitate)**.

### Operatorul va executa comparația între două obiecte ale clasei voastre și va returna true doar dacă ambii parametri vor fi egali.

### Adăugați un constructor și creați 4 obiecte(2 egale între ele și 2 diferite), executați comparația cu ajutorul operatorului supraîncărcat și afișați rezultatele.

### **Program 6:**

### Creați o clasă în care veți avea doi parametri: valoare1 și valoare2.

### Supraîncărcați **operatorul true** ca să returneze adevăr atunci când ambele valori sunt egale, și fals când nu.

### Adăugați un constructor și creați 2 obiecte(unul cu valori egale, altul cu diferite)

### Afișați rezultatul ca valoare bool.

### **Program 7:**

### Creați o clasă în care veți avea trei parametri de tip int.

### Supraîncărcați **conversia implicită și explicită float sau double(la alegere)** ca să returneze **rădăcina pătrată din (a+b+c)3.**

### Adăugați un constructor și creați un obiect.

### Afișați rezultatul cu ajutorul conversiei explicite și implicite.

### 

### Modulul adițional. Introducere în WPF. Lucrul cu fișiere audio și imagine.

#### Ce este WPF? Descriere, structura și elemente de bază.

### Ce este WPF?

### WPF sau Windows Presentation Foundation face parte din framework-ul .NET și are scopul de a oferi dezvoltatorilor un instrument comod și cu o varietate de posibilități pentru a crea aplicații pentru Windows.

### Versiunea inițială a fost introdusă în 2007, dar este actualizat consistent, ultima actualizare - Februarie 2020.

### WPF permite de a interacționa cu UI, documente, imagini, sunete, video și alte elemente în interiorul unei aplicații Windows.

Versiunea inițială a WPF era foarte similară cu Windows Forms, dar există un element foarte important care le diferențiază. WPF este creat pentru a interacționa cu DirectX și memoria grafică a calculatorului, în timp ce WinForms folosește doar procesorul și memoria operativă. Această schimbare devine foarte importantă atunci când avem nevoie de a construi aplicații cu o interfață grafică mai complexă care conține o multitudine de elemente grafice, video, animații etc.

Structura WPF

### WPF este structurat în două părți: front-end și back-end.

**Back-endu**l are ca limbaj de bază C# ceea ce permite crearea aplicațiilor cu utilizarea completă a avantajelor platformei .NET. C# este utilizat deci pentru scrierea tuturor funcțiilor care stau în spatele interfeței grafice și permit funcționarea ei.

**Front-endu**l sau interfața grafică a WPF utilizează XAML, care este un limbaj crea de către Microsoft pentru a crea interfețele aplicațiilor Windows și este similar cu XML, HTML și CSS.

Un alt avantaj al WPF este că acesta permit aranjarea elementelor UI în formă dinamică(ca o pagină WEB), spre deosebire de WinForms care are o structura mult mai rigidă.

Pentru a demonstra funcționalitatea unei aplicații simple care va porni, pauza și opri un fișier audio și va modifica imaginea de pe ecran în dependență de butonul apăsat.

Exemplu

namespace WpfApp1

{

public partial class MainWindow : Window

{

private MediaPlayer mediaPlayer = new MediaPlayer();

public bool opened;

public MainWindow()

{

InitializeComponent();

Imagine.Source = new BitmapImage(new Uri(@"D:\2.png"));

}

#region Butoane

private void Play\_Click(object sender, RoutedEventArgs e)

{

//Play sound

if (!opened)

{

mediaPlayer.Open(new Uri(@"D:\Enya – Only Time.mp3"));

opened = true;

}

mediaPlayer.Play();

Imagine.Source = new BitmapImage(new Uri(@"D:\play.png"));

}

private void Pause\_Click(object sender, RoutedEventArgs e)

{

mediaPlayer.Pause();

Imagine.Source = new BitmapImage(new Uri(@"D:\pause.png"));

}

private void Stop\_Click(object sender, RoutedEventArgs e)

{

mediaPlayer.Stop();

opened = false;

Imagine.Source = new BitmapImage(new Uri(@"D:\2.png"));

}

#endregion

}

}