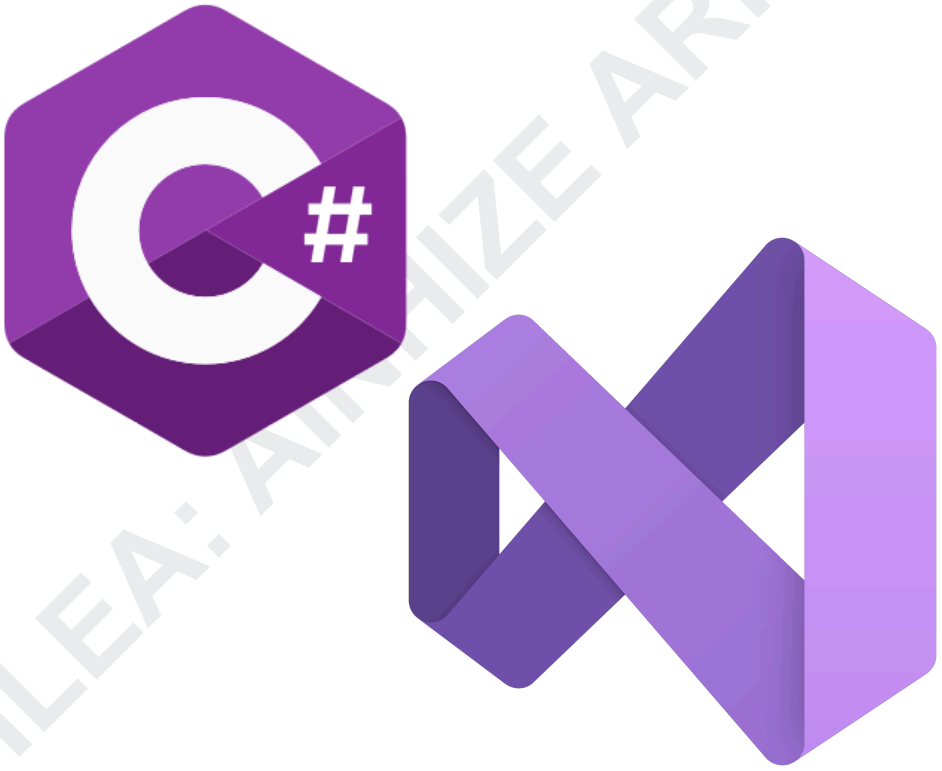
C#

INTERFAZEEN GARAPENA



AURKIBIDEA

1.- Sarrera

1.1- Visual Studio eta WinForms instalazioa

- C# .NET aplikazioak garatzeko, **Visual Studio 2022 Community** instalatzea gomendatzen da.
- Windows Forms aplikazioak sortzeko beharrezko tresnak bertan daude integratuta.

- Instalatzeko esteka: https://visualstudio.microsoft.com/es/vs/

1.2- Kaixo Mundua adibidea (Windows Forms aplikazioa)

- Adibide klasiko bat da Kaixo Mundua mezua agertzea aplikazio grafiko batean. WinForms aplikazio batean botoi bat gehitzen dugu eta sakatzean mezu bat agertzen da.

```
namespace KaixoMunduaApp
{
    public partial class Form1 : Form
    {
        public Form1(){
            InitializeComponent();
        }

        private void btnKaixo_Click(object sender, EventArgs e){
            MessageBox.Show("Kaixo Mundua!");
        }
    }
}
```

MessageBox.Show() → funtzioak testu-kodroa bat erakusten du pantailan

1.3- Programazio urratsak

- C# aplikazio bat garatzeko jarraitu beharreko urratsak:
  - ***Kodea idatzi:*** adibidez Form1.cs fitxategian.
  - **Konpilatu:** Visual Studio-k automatikoki egiten du (proiektua eraiki).
  - **Exekutatu:** aplikazioa martxan jarri (F5 edo Run botoia).

Ariketa txikia:

- Sortu Windows Forms aplikazio bat.
- Gehitu botoi bat formularioan.
- Klik egitean "Ongi etorri C#-ra" mezua ager dadila.

2.- Oinarrizko kontzeptuak

2.1- Izenen eremuak, aldagaiak eta motak

- **Aldagaiak datuak gordetzeko erabiltzen dira; eta motek, zer balio gorde daitekeen adierazten dute:**
  - int → zenbaki osoak.
  - float → zenbaki hamartarrak (memoria gutxiago)
  - double → zenbaki hamartarrak (zehaztasun handiagoa)
  - string → testuak
  - char → karaktere bakarrak.
  - bool → egia/faltsua

#Adb.:

```
int adina = 20;
float prezioa = 12.5f;
double altura = 172.75;
string mezua = "Kaixo!";
bool aktibo = true;
```

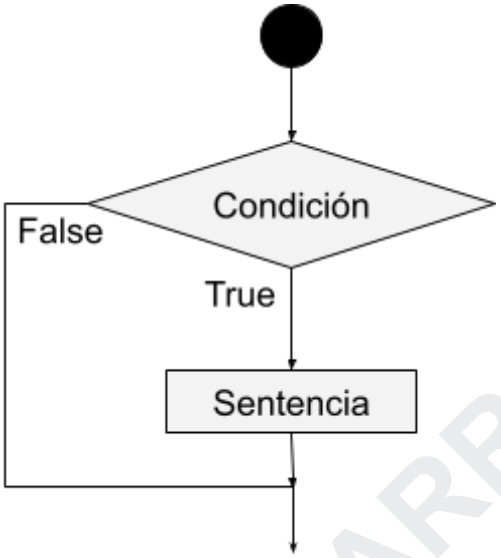
2.2- Baldintzazko egiturak (IF, ELSE IF, ELSE)

- **Baldintzazko egiturek programaren exekuzioaren fluxua kontrolatzen dute, baldintza bat betetzen den edo ez den arabera.**

- C#-ek baldintza mota nagusi hauek ditu, java-ren antzera:
  1.- **IF**
  2.- **IF -ELSE**
  3.- **IF -ELSE IF -ELSE**

2.2.1- IF

- **Eskema:**



- **Adibidea:**

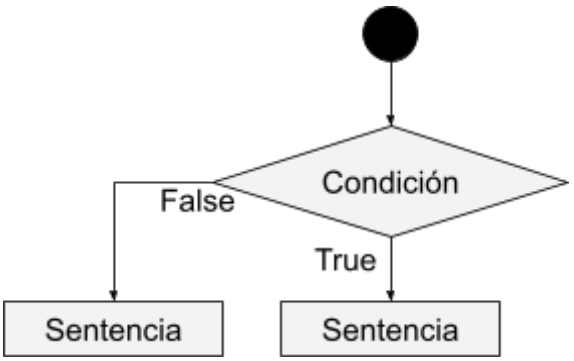
```
int x = 12, y = 5, z = 0;

if(y!=0){
    z = x/y;
}

Console.WriteLine("Ekuazioaren emaitza: " + z);
```

2.2.2- IF - ELSE

- **Eskema:**



- **Adibidea:**

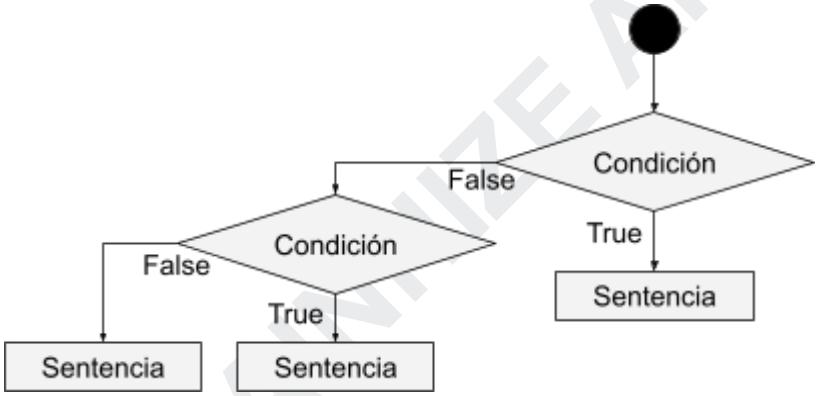
```
int x = 12, y = 5, z = 0;

if (y != 0){
    z = x / y;
}else{
    z = x - y;
}

Console.WriteLine("Ekuazioaren emaitza: " + z);
```

2.2.3- IF - ELSE IF - ELSE

- **Eskema:**



- **Adibidea:**

```
int num = -5;

if (num > 0){
    Console.WriteLine("Zenbakia positiboa da");
}else if (num == 0){
    Console.WriteLine("Zenbakia zero da");
}else{
    Console.WriteLine("Zenbakia negatiboa da");
}
```

2.3- Egitura errepikakorrak (for, foreach, while, do...while)

• Bucle edo errepikapen-egiturak kode zati bat behin eta berriro exekutatzeko erabiltzen dira.

• C#-en mota nagusiak hauek dira:

1.- While: Baldintza betetzen den bitartean.

2.- Do... while: Behin gutxienez exekutatzen da, eta gero baldintza berriz aztertzen da.

3.- For: Kontagailu baten bidez..

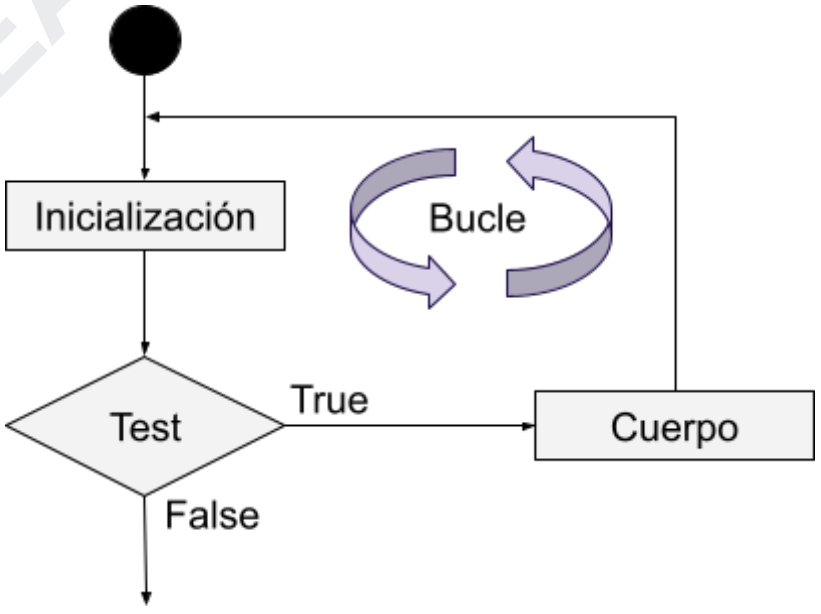
4.- Foreach: Array edo zerrenda bateko elementu guztietarako

ESTRUCTURA			

`inicialización;`
`while (condición){`
`  cuerpoDelBucle;`
`}`
`declaración siguiente;`
	`inicialización;`
	`do{`
	`  cuerpoDelBucle;`
	`}while(condición);`
	`decla. siguiente;`
		`foreach(`
		`var elementua in`
		`bilduma){`
		`  cuerpoDelBucle;`
		`}`
			`for(inic.;trem.;incr.)`
			`{`
			`  cuerpoDelBucle;`
			`}`
			`declaración siguiente;`

2.3.1- While

• Eskema:



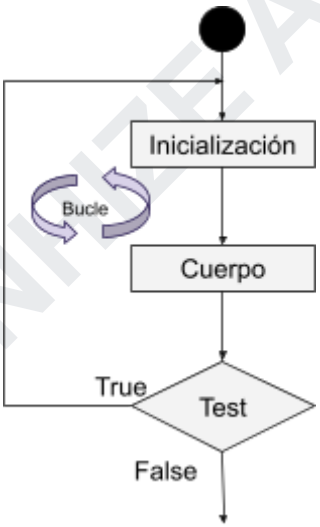
- **Adibidea:**

```
int kont = 0;

while (kont < 3)
{
    Console.WriteLine("Kont = " + kont);
    kont++;
}
```

2.3.2- do... While

- **Eskema:**



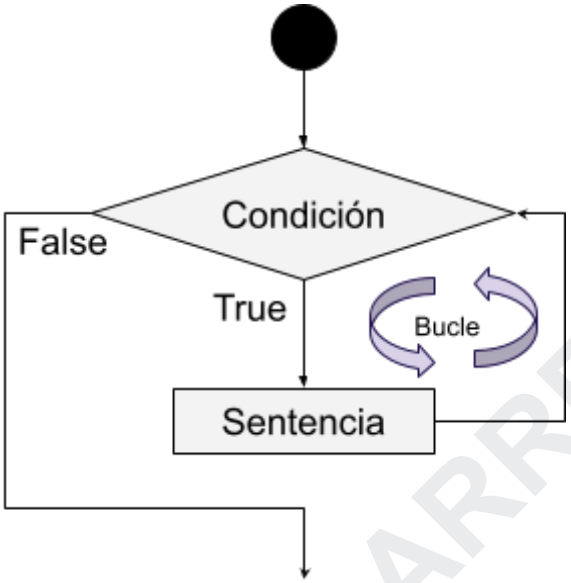
- **Adibidea:**

```
int n = 0;

do
{
    Console.WriteLine("n balioa: " + n);
    n++;
}
while (n < 2);
```

2.3.3- For

- **Eskema:**

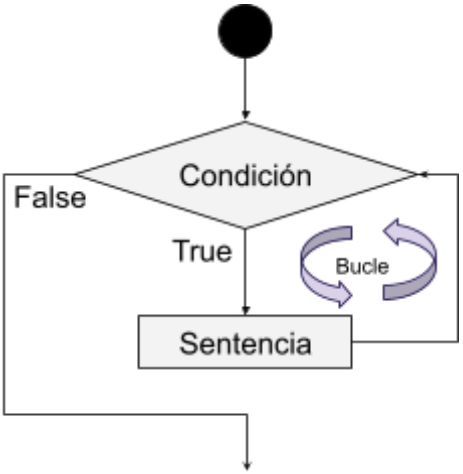


- **Adibidea:**

```
for (int i = 1; i <= 5; i++)
{
    Console.WriteLine("Zenbakia: " + i);
}
```

2.3.4- Foreach

- **Eskema:**



- **Adibidea:**

```
string[] izenak = { "Ane", "Jon", "Mikel" };

foreach (string izena in izenak)
{
    Console.WriteLine("Kaixo " + izena);
}
```

2.4- Erroreen kontrola (try, catch, finally)

- **Programa batean akatsak edo egoera ezustekoak gerta daitezke (adibidez, erabiltzaileak zenbaki baten ordez testua sartzen duenean). Horrelako egoerak kudeatzeko, C#-ek try-catch-finally egitura eskaintzen du.**

- **Eskema:**

```
try{
    // arriskutsua izan daitekeen kodea
}catch (Exception ex){
    // errorea gertatzean exekutatzen den kodea
    Console.WriteLine("Errorea: " + ex.Message);
}finally{
    // beti exekutatzen den kodea (aukerakoa)
}
```

- **Adibidea:**

```
try{
    Console.Write("Sartu zenbaki bat: ");
    int zenbakia = int.Parse(Console.ReadLine());
    Console.WriteLine("Sartutako zenbakia: " + zenbakia);
}catch (FormatException){
    Console.WriteLine("Errorea: zenbaki bat sartu behar da.");
}finally{
    Console.WriteLine("Programa amaitu da.");
}
```

2.4.1- Exception motak

Exception motak	Zertarako balio dute?
Exception	Orokorreko salbuespena; ez du zehazki zer gertatu den adierazten, baina guztiak harrapa ditzake.
DivideByZeroException	Zeroz zatitzen saiatzen zarenean jaurtitzen da. (adb. 1/0)
FormatException	Formatu okerreko datuak parseatzen saiatzen zarenean jaurtitzen da (adb. int.Parse("abc")).
ArgumentNullException	Metodo bati null argumentu ez onargarria pasatzen zaionean jaurtitzen da.
ArgumentOutOfRangeException	Metodo bati onargarriak ez diren argumentu bat pasatzen zaionean jaurtitzen da (adib. zerrenda baten indize okerra).
InvalidOperationException	Operazio bat ezinezkoa den egoera batean saiatzen zarenean jaurtitzen da (adib. kolekzioa iteratzen zauden bitartean aldatzen baduzu).
IOException	Fitxategiak edo sarrera/irteerako operazioak huts egiten duenean jaurtitzen da.
FileNotFoundException	Fitxategi bat aurkitu ez denean jaurtitzen da.
NullReferenceException	null-aren gainean metodo edo propietate bat erabiltzen saiatzen zarenean jaurtitzen da.
OverflowException	Zenbaki baten operazioak bere motako limiteak gainditzen dituenean jaurtitzen da.

3.- Objektuetara zuzendutako programazioa (POO)

• **OOP edo Objetuei Oinarritutako Programazioa,** programak datuak eta funtzioak objetu moduan antolatzen ditu, errepikapenak eta kudeaketa errazagoa ahalbidetuz.

3.1- Klaseak eta Objetuak

• **Klasea:** objetuak sortzeko plantilla edo molde bat da.
• **Objetua:** klase baten instantzia da, datu propioak eta funtzioak dituenak.

- **Sintaxia:**

```
class Pertsona
{
    public string izena;
    public int adina;
}

// Objektua sortzea
Pertsona p1 = new Pertsona();
p1.izena = "Ane";
p1.adina = 25;
```

3.2- Propietateak (get, set)

- **Propietateek aldagai pribatuak kontrolatuz irakurtzea eta idaztea ahalbidetzen dute.**
- **Get: irakurtzeko.**
- **Set: idazteko.**
- **Sintaxia:**

```
class Pertsona
{
    private int adina;

    public int Adina
    {
        get { return adina; }
        set { if (value >= 0) adina = value; }
    }
    public string Izena { get; set; } // Auto-implemented property
}

Pertsona p = new Pertsona();
p.Adina = 30;           // set
Console.WriteLine(p.Adina); // get
```

3.3- Metodoak

- **Metodoak klase baten funtzionalitateak definitzen dituzte, eta objetuek metodoak exekutatu ditzakete.**

- **Adibidez:**

```
class Kalkulagailua
{
    // Metodoa
    public int Batura(int a, int b)
    {
        return a + b;
    }
}

Kalkulagailua k = new Kalkulagailua();
Console.WriteLine(k.Batura(5, 3)); // 8
```

3.4- Konstruktoreak

- **Konstruktoreek klase bat instantziatzean, hasierazte dute (lo inicializan), ez dute void motarik eta izena klasearen berdina izan behar du.**

- **Adibidez:**

```
class Pertsona
{
    public string Izena { get; set; }
    public int Adina { get; set; }

    // Konstruktorra
    public Pertsona(string izena, int adina)
    {
        Izena = izena;
        Adina = adina;
    }
}
Pertsona p = new Pertsona("Jon", 28);
Console.WriteLine(p.izena);
```

3.5- Klase eta metodo estatikoak

• Estatiko den elementu bat klaseari dagozkio, ez objektu bakoitzari. Hau da, ez du behar objektu berri bat sortzea (new) funtzio edo datua erabiltzeko. Garrantzitsua da jakitea klase estatiko bat **globala** dela esan daitekeela, baina C#-en barruan.

• Klase estatikoak static hitza erabiliz deklaratzen dira. Ez dute instantzia izan daitezkeen objekturik, eta atributu estatikoak eta metodo estatikoak soilik izan ditzakete. Exekuzioa programaren hasieran edo lehen erabileran hasten da, eta ez dago objektu berririk sortu beharrik.

• Metodo estatikoak klasean bertan funtzionatzen dute, objekturik sortu gabe. Horretarako, static hitza erabiltzen da. ***Garrantzitsua***: metodo estatikoek ezin dute this edo klasearen ez-estatiko atributuak erabili, objektu bakoitzari dagokion datua ez dutelako.

• **Adibidez:**

```
static class Matematika{
    public static double Pi = 3.14159;

    public static double ZirkuluarenAzalera(double r){
        return Pi * r * r;
    }
}
class Kalkulagailua{
    public static int Batura(int a, int b){
        return a + b;
    }

    public int BaturaEzEstatikoa(int a, int b){
        return a + b;
    }
}
// Metodo estatikoaren erabilera:
Console.WriteLine(Kalkulagailua.Batura(4, 6)); // 10
// Metodo ez-estatikoaren erabilera:
Kalkulagailua k = new Kalkulagailua();
Console.WriteLine(k.BaturaEzEstatikoa(4, 6)); // 10
```

• **Laburbilduz:**

Ezaugarria	Estatikoa	Ez-Estatikoa
Objetuak behar?	ez	bai
this erabili daiteke?	ez	bai
Kudeaketa	klase mailako baliabideak	Objetu mailako baliabideak
Adibidea	Math.Pow()	new Kalkulagailua.batu()

3.6- Herentzia eta Polimorfismoa

```
    Animal (Klase Nagusia)
   ---------------------
   + Jan()
   + SoinuEgin()  <- virtual
           |
   ---------------
   |             |
 Txakur          Katua
+ KorrikaEgin()  + MiauEgin()
+ SoinuEgin()    + SoinuEgin()  <- override
```

• **Herentzia:** Txakur eta Katu klaseek Animal-itik metodoak eta atributuak jasotzen dituzte.
• **Polimorfismoa:** SoinuEgin() metodo bera, klase desberdinetan modu desberdinean exekutatzen da (override bidez).

3.6.1- Herentziak:

• Herentziak programazio objetuetara zuzentzean, oinarrizko kontzeptu bat da. Klase batek (subklaseak) beste klase baten (superklasearen) propietateak eta metodoak hereda ditzake. Kodearen berrerabilpena eta antolaketa erraztea du helburu.

```
class Pertsona
{
    public string Izena { get; set; }
    public int Adina { get; set; }

    public virtual void Erakutsi()
    {
        Console.WriteLine($"Izena: {Izena}, Adina: {Adina}");
    }
}

class Ikaslea : Pertsona // Ikaslea klaseak, Pertsona klasea heredatzen du
{
    public string Ikasgai { get; set; }

    public override void Erakutsi()
    {
        Console.WriteLine($"Izena: {Izena}, Adina: {Adina}, Ikasgai:
{Ikasgai}");
    }
}
```

3.6.1- Polimorfismoa:

- Polimorfismoak metodo berinak objetu desberdinetan portaera ezberdina izatean datza. Horretarako, superklaseko metodoak Virtual gisa definitzen dira, eta subklaseetan override erabiliz berridazten dira.

```
class Programa{
    static void Main(string[] args){
        Pertsona p1 = new Pertsona { Izena = "Jon", Adina = 30 };
        Pertsona p2 = new Ikaslea { Izena = "Ane", Adina = 20, Ikasgai =
"Programazioa" };
        p1.Erakutsi();// Output: Izena: Jon, Adina: 30
        p2.Erakutsi();// Output: Izena: Ane, Adina: 20, Ikasgai:
Programazioa
    }
}
```

3.7- Klaseak eta Metodo birtualak

• Metodo bat **virtual** gisa definitzen bada, subklaseetan onverride erabiliz berridatze daiteke. Horrek **polimorfismoa**, hau da, izen bereko metodoek objetuaren arabera portaera desberdinak izatea ahalbidetzen du.

```
class A {
    public virtual void Mezua(){
        Console.WriteLine("A klasetik datorren mezua");
    }
}

class B : A {
    public override void Mezua(){
        Console.WriteLine("B klasetik datorren mezua (override)");
    }
}

//adb. erabiltzerakoan:
class Programa{
    static void Main(string[] args){
        A objA = new A();
        objA.Mezua(); // Output: A klasetik datorren mezua

        B objB = new B();
        objB.Mezua(); // Output: B klasetik datorren mezua (override)

        // Polimorfismoa:
        A objPoli = new B();
        objPoli.Mezua(); // Output: B klasetik datorren mezua (override)
    }
}
```

Puntu garrantzitsuak:

• virtual → superklasean erabiltzen da, metodoa berridazteko aukera ematen du.

• override → subklasean erabiltzen da metodoa berridazteko.

• base.metodoa() → Subklase batean, superklaseko metodoaren bertsioa exekutatzeko aukera ematen du. (JAVA-ko .super(...)-ren baliokidea da)

```
class A{
    public A(string izena){
        Console.WriteLine("A eraikitzailea: " + izena);
    }
}

class B : A{
    public B(string izena, int adina) : base(izena){
        Console.WriteLine("B eraikitzailea, adina: " + adina);
    }
}
```

3.8- Interfazeak

• **Zer da interfazea?** Metodoaren sinadura definitzen duen "kontratu" bat da, baina ez du inplementaziorik ematen. Klase batek interfazea inplementatzen duenean, bertan definitutako metodo guztiak bete behar ditu, polimorfismoa eta abstrakzioa erraztea du helburu.

• **Ezaugarriak:**

- Interfazea *interface* hitzarekin definitzen da.
- Sinadura metodoek soilik dute (gorputzik gabe).
- Klase batek interfazea inplementatzen duenean, metodo guztiak *public* moduan definitu behar ditu.
- Klase batek interfaze abt baino gehiago inplementatu ditzake. (C#-ek ez du herentzia anizkoitza onartzen, baina interfazeen bidez bai.)

```
interface IMugikorra{
    void Deitu(string zenbakia);
    void MezuaBidali(string mezua);
}
class Telefonoa : IMugikorra{
    public void Deitu(string zenbakia){
        Console.WriteLine("Deitzen " + zenbakia + "-ra...");
    }
    public void MezuaBidali(string mezua){
        Console.WriteLine("Mezua bidali da: " + mezua);
    }
}
```

```
//adb. erabiltzerakoan:
class Programa{
    static void Main(string[] args){
        IMugikorra mugikorra = new Telefonoa();
        mugikorra.Deitu("666111222");
        mugikorra.MezuaBidali("Kaixo, zer moduz?");
    }
}
```

4.- Datu Egiturak

- Datu egiturek informazioa gorde eta kudeatzeko modu desberdinak eskaintzen dituzte. C#-ek aukera asko ditu, baina hasieran erabilienak hauek dira: array-ak eta zerrendak (List<L>).

4.1- Array-ak

- Array-ek elementu multzoak gordetzen dituzte, hauek tamaina finkoa dute eta bertan indize bidez sar daitezke elementuak (0-tik hasita).

```
// Deklarazioa
datu_mota[] izena = new datu_mota[tamaina];

// Iniciar el array
int[] zenbakiak = { 1, 2, 3, 4, 5 };

//ADB.
int[] zenbakiak = new int[3];
zenbakiak[0] = 10;
zenbakiak[1] = 20;
zenbakiak[2] = 30;

for (int i = 0; i < zenbakiak.Length; i++){
    Console.WriteLine("Balioa: " + zenbakiak[i]);
}
```

4.2- Zerrendak (List<L>)

• Datu egitura dinamikoak dira, eta elementu kopurua aldakorra izan daiteke. Memoriaren kudeaketa automatikoki egiten dute, eta horregatik array tradizionalak baino malguagoak dira.

• C#-en, zerrendak *System.Collections.Generic* espazioko *List<L>* klasearen bidez erabiltzen dira.

```
//Zerrendak:
List<datu_mota> izena = new List<datu_mota>();

//ADB.
using System;
using System.Collections.Generic;
class Programa{
    static void Main(string[] args){
        List<string> izenak = new List<string>();
        izenak.Add("Ane");
        izenak.Add("Jon");
        izenak.Add("Mikel");
        foreach (string izena in izenak){
            Console.WriteLine("Kaixo " + izena);
        }
        Console.WriteLine("Guztira: " + izenak.Count + " elementu");
    }
}
```

4.3- Beste egitura batzuk

• C#-ek array eta zerrendez gain beste datu egitura erabilgarri batzuk ere eskaintzen ditu. Adibidez, *Dictionary<K,V>* egitura erabiltzen da gako-balio bikoteak gordetzeko, bilaketa azkarrak egiteko oso erabilgarria dena.

• Bestalde, *Queue<T>* ilara baten logikan oinarritzen da (FIFO – First In, First Out), hau da, lehen sartzen den elementua lehen ateratzen da. Azkenik, *Stack<T>* pila baten modura funtzionatzen du (LIFO – Last In, First Out), azken sartzen den elementua lehen irteten delarik.

```
Dictionary<int, string> ikasleak = new Dictionary<int, string>();

ikasleak.Add(1, "Ane");
ikasleak.Add(2, "Jon");

Console.WriteLine(ikasleak[1]);  // Output: Ane
foreach (var ikasle in ikasleak)
{
    Console.WriteLine($"ID: {ikasle.Key}, Izena: {ikasle.Value}");
}
```

5.- WinForms aplikazioak

• Windows Forms (***WinForms***) C#-en erabiltzen den teknologia bat da idazmahaiko aplikazio grafikoak sortzeko. Formulario batek hainbat kontrol izan ditzake: botoiak, testu-koadroak, etiketa testuak, etab. Kontrol bakoitzak bere ***eventuak*** ditu, eta horiei funtzio edo metodo bat lotuz gero, erabiltzailearen ekintzek eragindako kodea exekutatu daiteke.

• Adibidez, botoi bat sakatzean mezu bat agertzea:

```
namespace WinFormsAdibidea{
    public partial class Form1 : Form{
        public Form1(){
            InitializeComponent();
        }
        private void btnKaixo_Click(object sender, EventArgs e){
            MessageBox.Show("Kaixo! Botoia sakatu duzu.");
        }
    }
}
```

• Horrez gain, TextBox baten bidez erabiltzaileari datuak eska dakizkioke, eta gero sartutakoa balidatu daiteke, adibidez zenbaki bat den egiaztatuz:

```
private void btnBidali_Click(object sender, EventArgs e)
{
    if (int.TryParse(txtZenbakia.Text, out int zenbakia)){
        MessageBox.Show("Sartutako zenbakia: " + zenbakia);
```

```
    }else{
        MessageBox.Show("Errorea: zenbaki bat sartu behar da.");
    }
}
```

• Bestalde, formulario batetik bestera informazioa pasatzea ere posible da parametroen bidez:

```
Form2 bigarrenFormularioa = new Form2(txtIzena.Text);
bigarrenFormularioa.Show();
```

• Eta Form2-ko eraikitzailean jasotzen da parametroa:

```
public Form2(string izena){
    InitializeComponent();
    lblIzena.Text = "Kaixo " + izena;
}
```

6.- DLL-ak Sortzen eta Aplikazioetan gehitzen

• C# .NET ingurunean DDL fitxategiak (Dynamic Link Library) erabiltzen dira kode modularrak eta berrerabilgarriak sortzeko. DDl batek klaseak, metodoak edo kontrol pertsonalizatuak eduki ditzake, eta hainbat proiektutan erabil daitezke erreferentzia gehituz.

6.1- DLL bat sortzea (Kaixo adibidea)

1.- Visual Studio-n proiektu berria sortu: *"Blibioteca de controles de Windows Form (.NET Framework)"* txantiloia aukeratu. (adib. KaixoDll izena eman)

2.- Fitxategi nagusiari izena aldatu: *Class1.cs → Kaixo.cs*

3.- Kode hau gehitu:

```
namespace KaixoDll
{
    public class Kaixo
```

```
    {
        public string MezuaIkusi()
        {
            return "Kaixo Mundua";
        }
    }
}
```

4.- Konpiltu proiektua (*ctrl + shift + B*), eta visual studio-k *KaixoDll.dll* bezala sortuko du *\bin\debug* karpeta barruan.

6.2- DLL hori aplikazio batean erabiltzen (KaixoForms)

1.- Sortu beste proiektu bat: *"Aplikacion de Windoejws Forms (.NET Framework)"* → KaixoForms

2.- Formularioan botoi bat jarri (btnMezuaErakutzi).

3.- Gehitu aurreko DLL-aren erreferentzia:

Proyecto → *Agregar referencia* → *Examinar* → hautatu *KaixoDll.dll*

4.- Gehitu *using Kaixodll;* kodearen goialdean, eta botoiaren click ebentuan hau jarri:

```
private void btnMezuaErakutzi_Click(object sender, EventArgs e)
{
    Kaixo kaixo = new Kaixo();
    MessageBox.Show(kaixo.MezuaIkusi());
}
```

- Honen ondoren aplikazioko botoia zakatzean, mezu kutxa barruan DLL-ko funtzioan esandako mezua agertuko da.

6.3- DLL pertsonalizatuak: erabiltzaile-kontrolak

• DLL batean UserControl bat sortuz, kontrol grafiko berriak (botoiak, taulak, grafikoak...) sor daitezke. Kontrol horiek beste aplikazioetan erabil daitezke tresna-panelera gehituta.

6.4- Adibidea: Grafikoa kontrol pertsonalizatua

1.- Sortu proiektu berria: *"Biblioteca de controles de Windows Forms (.NET Framework)"* → izena: GrafikoaDll.

2.- Fitxategiari izena aldatu: UserControl1.cs → Grafikoa.cs.

3.- Gehitu Chart kontrol bat eta pertsonalizatu:

- *BackColor* → DimGray
- *Titles* → Helmuga turistiko nagusiak
- *Series.ChartType* → Pie
- *Modifiers* → Public

4.- Konpilatu proiektua: *GrafikoaDll.dll* sortuko da.

5.- Beste proiektu batean gehitzeko:

- Toolbox → General → eskuin botoia → Choose Items...
- Examinar → hautatu *GrafikoaDll.dll*
- Eta zure kontrola tresna gisa erabilgarri egongo da.

6.5- Kontrol pertsonalizatua datuekin erabiltzea

```
private void Form1_Load(object sender, EventArgs e)
{
    List<Datua> datuak = new List<Datua>()
    {
        new Datua() { gakoa = 1, balioa = 24 },
        new Datua() { gakoa = 2, balioa = 31 },
        new Datua() { gakoa = 3, balioa = 63 }
    };

    foreach (var kontrola in kontrolak)
    {
        kontrola.Titles[0].Text = "HELMUGA TURISTIKO NAGUSIAK";
        kontrola.DataSource = datuak;
        kontrola.Series[0].YValueMembers = "balioa";
        kontrola.Series[0].XValueMember = "gakoa";
        kontrola.DataBind();
    }
}
```

- Eta Datua klasea:

```
internal class Datua
{
    public int gakoa { get; set; }
    public int balioa { get; set; }
}
```

6.6- Erabilera praktikoa

- DLL-ak erabiliz:
    - Komponenteak ***berrerabili*** daitezke proiektu askotan.
    - Kodea ***modularra*** eta ***mantentzeko errazagoa*** bihurtzen da.
    - Aldaketa bat DLL batean eginez, aplikazio guztiek bertsio berria erabil dezakete ***erreferentzia eguneratzean***.

7.- Aginte Koadroak eta ORM-ak (Datu baseekin konexioa)

• Aginte koadroak, datu multzoa modu grafikoan erakusteko tresnak dira. Datu-baseko informazioa irakurtzea eta erabiltzaileari modu argi zein bisualean aurkeztea dute helburutzat. Adibidez, grafikoen bidez.

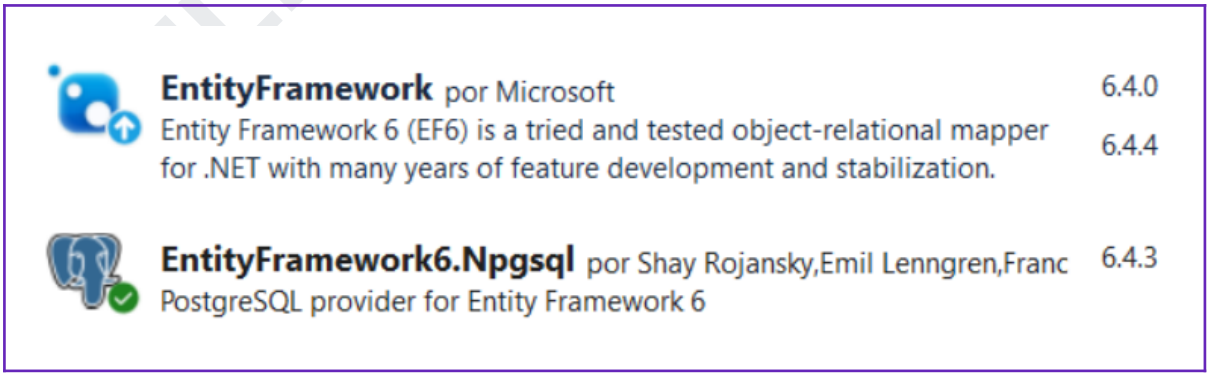
• Horretarako, apunte hauetan hurrengo hau erabiliko dugu:
  • **C# .NET (WinForms)** ← interfazea sortzeko
  • **Entity Framework (ORM)** ← datu-basearekin lan egiteko.
  • **PostgreSQL** ← datu-basea, datuak gordetzeko.

7.1- Entity Framework eta ORM kontzeptua

• ***Entity Framework (EF)***, ***ORM (Object Relational Mapper)*** baten adibide da. ***ORM*** batek programatzaileari aukera ematen dio datu-basearekin *objetuen bidez* lan egiteko, SQL kontsultak eskuz idatzi beharrik gabe. Horrela, programatzaileak aplikazioaren logikan zentratu daitezke, eta ez datu-baseko egitura zehatzetan.

• EF-k (***Entity Framework***) hainbat datu-base onartzen ditu lan egiteko orduan: SQL Server, MySQL, Oracle, PostgreSQL, etab.

• Gure kasuan PostgreSQL erabiliko dugu, eta horretarako ***NuGet*** bidez instalatuko ditugu hurrengo bi paketa hauek: ***EntityFramework*** eta ***EntityFramework6.Npgsql***. Instalazioa egiteko, ***WindForm*** proiektu barruan, ***Proyecto → Administrar paquetes NuGet*** barruan egingo dugu.



7.2- PostgreSQL datu-basea prestatzen

1.- PgAdmin ireki eta SalmentaDB ieneko datu-basea sortu.

2.- Bertan hurrendo taulak sortu eta erlazionatuko ditugu:

- ***Taulak:*** Bezero, Saltzailea, Salmenta.
- ***Eralazioa*:**
  - ⟩⟩ Saltzaile 1 → bezero asko izan ditzake.
  - ⟩⟩ Bezero 1 → salmenta asko izan ditzake.

• Datu-basearen egitura automatikoki inporta daideke .sql fitxategi baten laguntzaz (gure kasuan Moodle-n dagoen SalmentaDb.sql erabiliko dugu).

Tools → Query Tool → Open File → Execute

• Ondoren, egitura egiaztatzeko: *taularen gainean* eskuin botoiarekin → *View/Edit Data* → *All Rows*.

7.3- Konexioaren konfigurazioa (App.config)

• Aplikazioak datu-basearekin konektatzeko, *App.config* fitxategian sarrera bat gehitu beharko dugu:

```
<connectionStrings>
  <add name="SalmentaDbContext"
connectionString="Server=localhost;port=5432;Database=SalmentaDB;Username=openpg;Password=
openpgpwd;"
       providerName="Npgsql" />
</connectionStrings>
<entityFramework>
  <providers>
    <provider invariantName="System.Data.SqlClient"
              type="System.Data.Entity.SqlServer.SqlProviderServices,
EntityFramework.SqlServer" />
    <provider invariantName="Npgsql"
              type="Npgsql.NpgsqlServices, EntityFramework6.Npgsql" />
  </providers>
  <defaultConnectionFactory type="Npgsql.NpgsqlConnectionFactory, EntityFramework6.Npgsql"
/>
</entityFramework>
```

• *Username* eta *Password* parametroak zure PostgreSQL konfigurazioaren arabera egokitu behar dira (OpenERP/Odoo-k sortutako "openpg" erabiltzailea eta "openpgpwd" pasahitza erabiltzen dira normalean).

7.4- Ereduak eta DbContext klasea

• Models karpetan (eskuz sortu ez baduzu karpeta dau) eta bertan hurrengo 4 fitxategiak sortuko ditugu:

SalmentaDbContext.cs:

```
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;

namespace AginteKoadroa.Models{
    public class SalmentaDbContext : DbContext{
        public SalmentaDbContext() : base("SalmentaDbContext") { }
        public DbSet<Bezeroa> Bezeroa { get; set; }
        public DbSet<Saltzailea> Saltzailea { get; set; }
        public DbSet<Salmenta> Salmenta { get; set; }
        protected override void OnModelCreating(DbModelBuilder modelBuilder){
            modelBuilder.HasDefaultSchema("public");
            modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
        }
    }
}
```

Bezeroa.cs:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace AginteKoadroa.Models{
    public partial class Bezeroa{
        [Key]
        public string Id { get; set; }
        public string Izena { get; set; }
        public string Helbidea { get; set; }
        public string Telf { get; set; }
        public string Emaila { get; set; }
```

```
        public string SaltzaileaId { get; set; }
        public virtual Saltzailea Saltzailea { get; set; }
        public virtual List<Salmenta> Salmenta { get; set; }
    }
}
```

Saltzailea.cs:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace AginteKoadroa.Models{
    public partial class Saltzailea{
        [Key]
        public string Id { get; set; }
        public string Izena { get; set; }
        public string Taldea { get; set; }

        public virtual List<Bezeroa> Bezeroa { get; set; }
    }
}
```

Salmenta.cs:

```
using System;
using System.ComponentModel.DataAnnotations;

namespace AginteKoadroa.Models{
    public partial class Salmenta{
        [Key]
        public int Id { get; set; }
        public DateTime Data { get; set; }
        public decimal Zenbatekoa { get; set; }

        public string BezeroaId { get; set; }
        public virtual Bezeroa Bezeroa { get; set; }
    }
}
```

7.5- LINQ bidezko datuen bistaratzea grafikoetan

- **Aplikazioaren formulario nagusian (Form1) Load ekitaldia erabiliko dugu datuak irakurri eta grafikoetan erakusteko. Hau aktibatzeko Form1-en double click eginda automatikoki zortzen da.**

Adib. 1 → Saltzaile bakoitzaren bezero kopurua:

```
private void Form1_Load(object sender, EventArgs e){
    using (var db = new SalmentaDbContext()){
        var bezeroaData = db.Bezeroa
            .Include("Saltzailea")
            .GroupBy(b => b.Saltzailea.Izena)
            .ToDictionary(g => g.Key, g => g.Count());

        if (bezeroaData != null && bezeroaData.Count > 0){
            chart1.DataSource = bezeroaData;
            chart1.Series[0].YValueMembers = "Value";
            chart1.Series[0].XValueMember = "Key";
            chart1.DataBind();
        }
    }
}
```

Adib. 2 → Salmentarik handienak dituzten bezeroak:

```
private void Form1_Load(object sender, EventArgs e){
    using (var db = new SalmentaDbContext()){
        var salmentaData = db.Salmenta
            .Include("Bezeroa")
            .GroupBy(b => b.Bezeroa.Izena)
            .ToDictionary(g => g.Key, g => g.Sum(b => b.Zenbatekoa));

        if (salmentaData != null && salmentaData.Count > 0){
            grafikoa1.KargatuDatuak(salmentaData);
        }
    }
}
```

7.6- Laguntza sistemak eta dokumentazioa

• Aplikazioari laguntza-sistema gehitzeko ***HelpProvider*** kontrola erabil daiteke. Honek ***.chm*** fitxategiak erabil ditzake erabiltzaileari azalpenak emateko.

1.- Formularioaren propietateak:

- ***HelpButton = True***
- ***MaximizeBox = False***
- ***MinimizeBox = False***

2.- ***HelpProvider*** kontrola gehitu eta grafiko bakoitzari testu laguntza bat ezarri:

- ***Abid.:*** "Saltzaile bakoitzak (Y ardatza) dituen bezeroak (X ardatza) erakusten dira."

• Beste aukera sinple bat ToolTip kontrola da: sagua elementu baten gainetik pasatzean testu labur bat bistaratzen du.

7.7- Aplikazioaren instalatzailea

• Aplikazioa banatzeko, ***Microsoft Installer (Setup Project)*** erabil daiteke Visual Studio-n.

1.- Gehitu proiektu berria soluzio berean: ***Setup Project*** motakoa.

2.- "Application Folder"-ean gehitu proiektuaren Output.

3.- Sortu bi sarbide zuzen::

- ***Desktop***
- ***Programs Menu*** → ***Applicaciones***

4.- Gehitu ikono bat eta lotu bi sarbideetan.

5.- ***Setup1*** proiektuan:

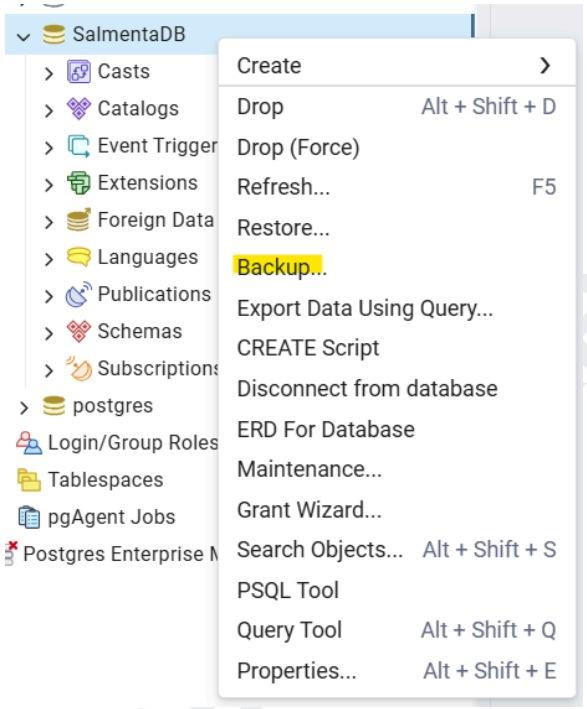
- ***TargetPlataform*** → x64.
- ***AddRemoveProgramsIcon*** aktibatu.

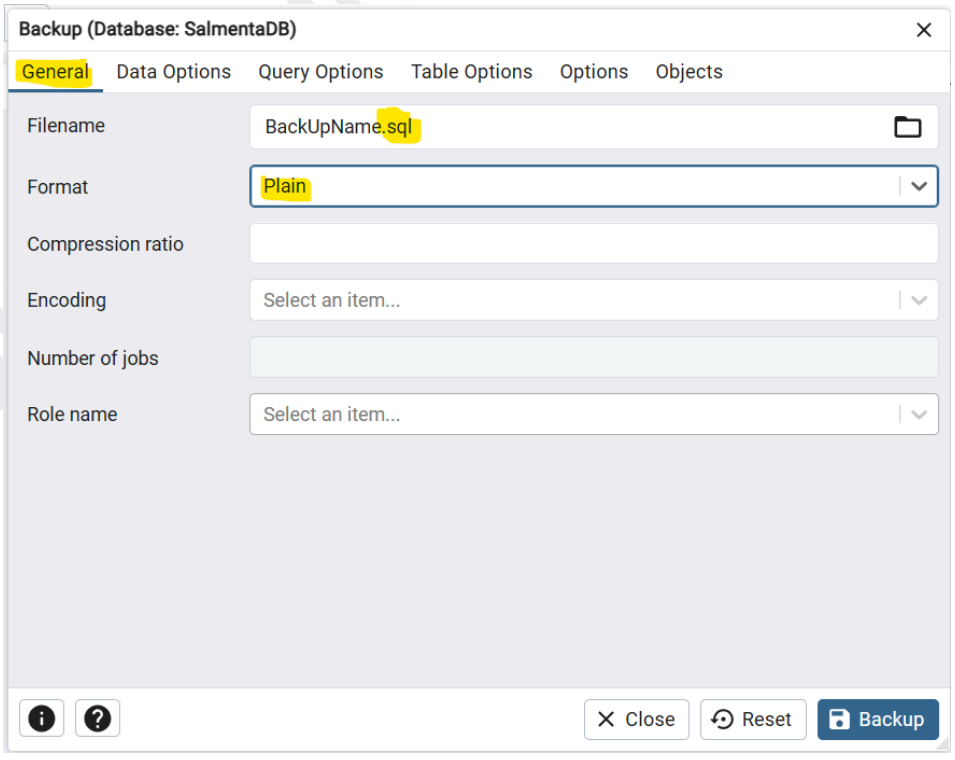
6.- "Prerequisitos" egiaztatu eta konfigurazioa ***Release*** modura jarri.

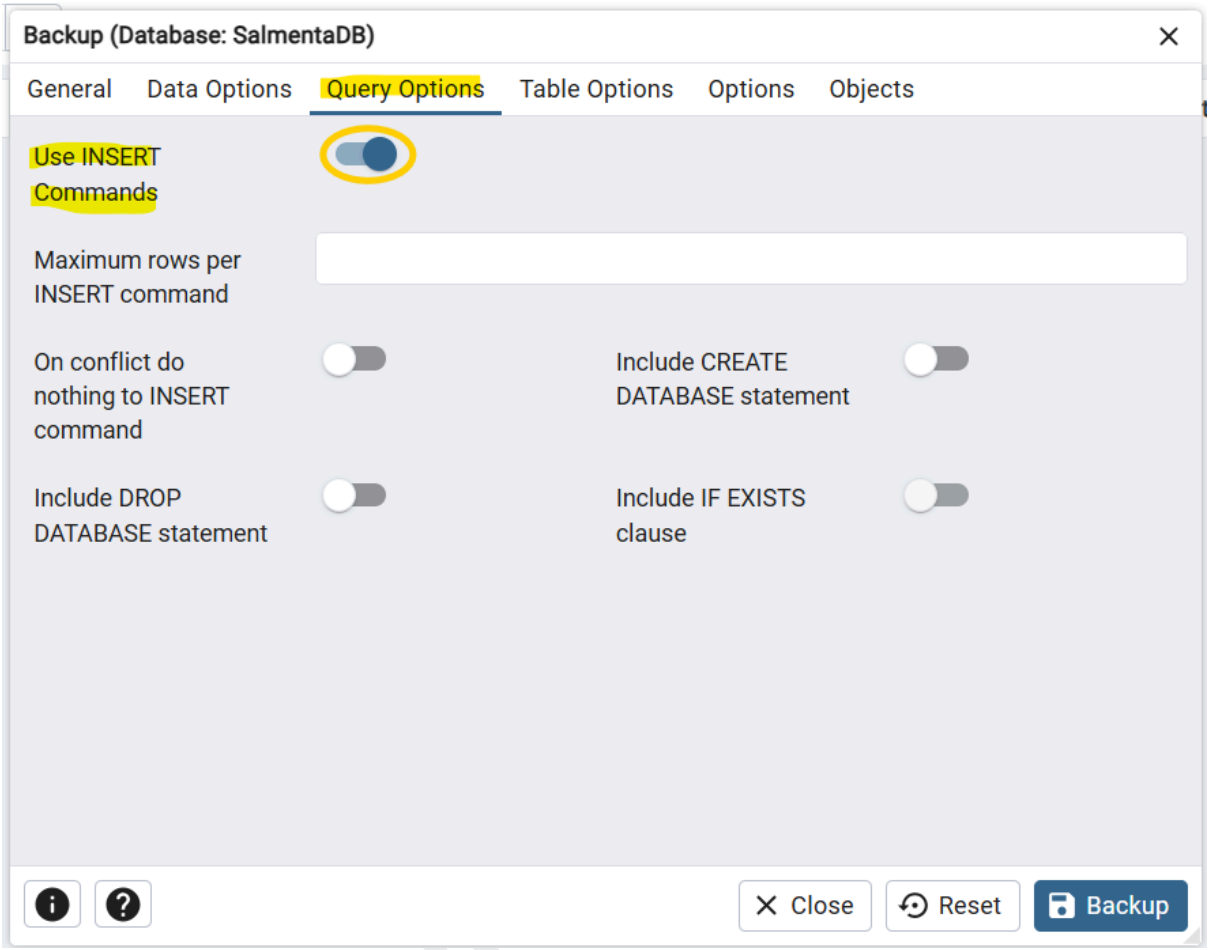
7.- Konpilatu: instalatzailea automatikoki sortuko da ***\bin\Release*** karpetan.

7.8- Babeskopiak

• Datu-basearen backup pgAdmin4: **Eskuineko botoia → Backup → fileName: XXX.sql → Format: Plain → Query options: "Insert command" → Objects: select all**







• Modu honetara, fitxategia Documents karpetan gordeko da, beste leku batean gorddetzeko, file name atalean gordenahi den ruta absolutua eta ditzategiaren izena idatzi beharko da. Adibidez:

C:\Users\<usename>\Downloads>\FileBackUp.sql