FUNDAMENTAL .NET

PEMBERDAYAAN UMAT BERKELANJUTAN

C#

CONTROLS
CLASS
INTERFACES
CONSTRUCTORS
OPERATORS
OOP
BASICS
FUNCTIONS
OVERLOADING
ARRAYS
INHERITANCE
STRINGS
OBJECT

PENYUSUN

YOGA HENDRAPRATAMA
AF'IDATUL MAGHFIROH
ANNISA RIZKY MULYA

TAHUN 2021

Visual Studio

BAB I

PENGENALAN

1.1 Bahasa Pemrograman C#

C# adalah bahasa pemrograman berbasis obyek modern yang dikembangkan pada tahun 2000 oleh Anders Hejlsber di Microsoft sebagai rival pemrograman Java. Pada saat itu, Sun (sekarang Oracle) tidak ingin Microsoft melakukan perubahan pada Java, sehingga Microsoft memutuskan untuk mengembangkan bahasa pemrogramannya sendiri. Sejak pertama kali dikembangkan, dengan dukungan yang luas dari Microsoft, C# telah berkembang dengan sangat cepat. Sekarang C# adalah salah satu dari bahasa pemrograman paling populer di dunia.

C# atau yang dibaca C sharp adalah bahasa pemrograman sederhana yang digunakan untuk tujuan umum, dalam artian bahasa pemrograman ini dapat digunakan untuk berbagai fungsi misalnya untuk pemrograman server-side pada website, membangun aplikasi desktop ataupun mobile, pemrograman game dan sebagainya. Selain itu C# juga bahasa pemrograman yang berorientasi objek, jadi C# juga mengusung konsep objek seperti inheritance, class, polymorphism dan encapsulation.

Dalam prakteknya C# sangat bergantung dengan framework yang disebut .NET Framework, framework inilah yang nanti digunakan untuk mengcompile dan menjalankan kode C#. C# dikembangkan oleh Microsoft dengan merekrut Anders Helsberg. Tujuan dibangunnya C# adalah sebagai bahasa pemrograman utama dalam lingkungan .NET Framework. Banyak pihak juga yang menganggap bahwa Java dengan C# saling bersaing, bahkan ada juga yang menyatakan jika pernah belajar Java maka belajar C# akan sangat mudah dan begitu juga sebaliknya. Anggapan tersebut sebenarnya tidak salah karena perlu diketahui sebelum adanya C# Microsoft mengembangkan J++ dengan maksud mencoba membuat Java agar berjalan pada

platform Windows, karena adanya masalah dari pihak luar maka Microsoft menghentikan proyek J++ dan beralih untuk mengembangkan bahasa baru yaitu C#.

1.2 Apa Itu .Net Framework

Net Framework adalah sebuah perangkat lunak kerangka kerja yang berjalan utamanya pada sistem operasi Microsoft Windows, Kerangka kerja ini menyediakan sejumlah besar pustaka pemrograman komputer dan mendukung beberapa bahasa pemrograman sehingga memungkinkan bahasa-bahasa tersebut berfungsi satu dengan lain dalam pengembangan sistem serta mampu meminimalkan proses dalam pembangunan suatu aplikasi atau suatu sistem informasi dalam linkungan terdistribusi internet.

Tujuan .Net Framework:

- Menyediakan lingkungan pemograman berorientasi objek atau Object Oriented Programming (OOP) yang konsisten meskipun kode objek disimpan dan dijalankan secara lokal tetapi dapat disebarkan melalui internet dan di jalankan secara remote (di jalankan dari suatu tempat).
- Menyediakan lingkungan untuk menjalankan suatu kode yang dapat mengeliminasi masalah performa dari lingkungan scripted (halaman).
- Menyediakan lingkungan untuk menjalankan suatu kode yang menjamin keamanan saat kode di jalankan, termasuk kode yang di buat oleh pihak yang tidak di ketahui/ pihak ketiga yang setengah dipercaya.

.NET Framework terdiri dari dua buah bagian utama, yaitu Common Language Runtime (CLR) dan Base Class Library (BCL).

1. **Common Language Runtime (CLR)** adalah pondasi utama (bahasa umum) dari .NET Framework ,yang menjalankan aplikasi .NET Framework menyediakan sejumlah layanan pada berbagai hal seperti :

a. Pengaturan Memori

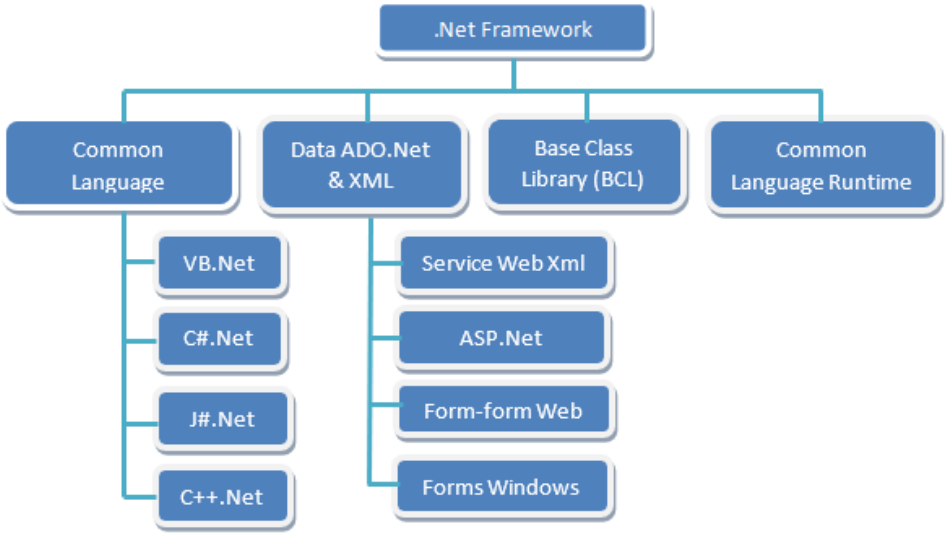
b. Mengelola kode (melakukan eksekusi kode)

c. Melakukan verifikasi terhadap keamanan kode

d. Menentukan hak akses dari kode, melakukan kompilasi kode, dan berbagai layanan system lainnya.

2. **Base Case Library (BCL)** bersifat berorientasi terhadap objek yang akan menyediakan jenis dari fungsi-fungsi pengaturan kode.

Seperti telah disebutkan sebelumnya bahwa .Net Framework memilki beberapa bahasa pemograman di dalamnya, termasuk VB.Net, maka struktur .Net Framewok paling utama dibangun oleh bahasa pemograman .Net. Dapat rekan-rekan lihat di bawah ini untuk struktur dasar .Net Framework;

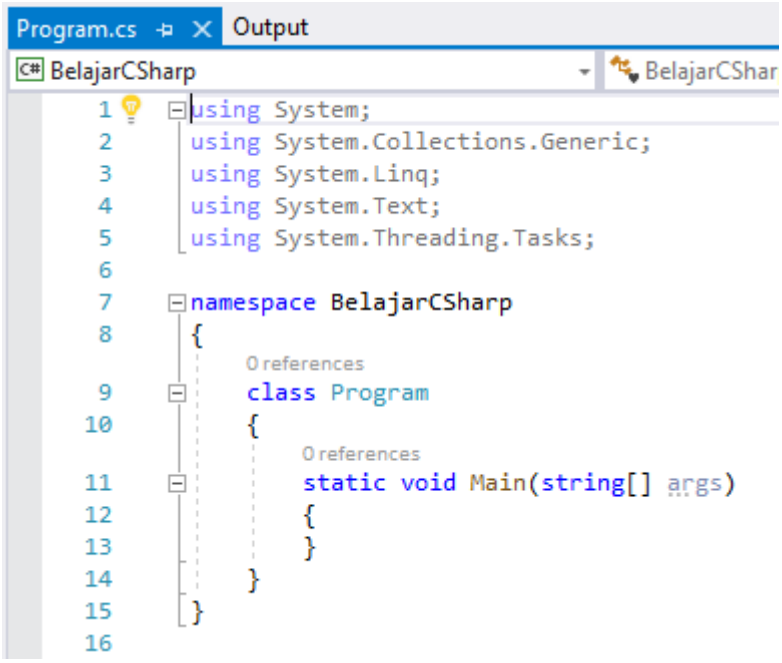


BAB II

BAHASA PEMROGRAMAN C#

2.1 Elemen Pemrograman C#

Seperti yang sudah Anda lihat, ketika Anda membuat aplikasi baru di Visual Studio, secara default, Visual Studio akan menyediakan beberapa elemen dasar untuk Anda. Elemen-elemen tersebut adalah namespace, class, dan method.



1. Namespace

.NET framework menggunakan namespace sebagai cara untuk memisahkan file-file class kedalam kategori yang terkait. Selain itu, penggunaan namespace juga bisa membantu menghindari bentroknya penamaan dalam aplikasi yang mungkin berisi class-class dengan nama yang sama.

Beberapa namespace yang ada dalam Bahasa pemrograman C#

.NET Namespace	Deskripsi
System	Menyediakan berbagai kelas yang berhubungan data, komputasi matematika, pembangkitan bilangan acak (Random), perubah lingkungan (runtime environment), serta penanganan memori menggunakan garbage collection.
System.Collections	Menyediakan kelas-kelas pemuat (container).
System.Collections.Generic	Menyediakan kelas-kelas pemuat (container) yang dapat di-custom-kan menggunakan tipe-tipe data generik.
System.Data	Namespace yang diperlukan untuk koneksi ke basis data bertipe relasional menggunakan ADO.NET.
System.Data.Odbc	
System.Data.OracleClient	
System.Data.Data.Oledb	
System.Data.SqlClient	
System.IO	Namespace yang berhubungan dengan asupan (input) dan keluaran (output), kompresi data, dan manipulasi port.
System.Compression	
System.Ports	
System.Drawing	Namespace yang diperlukan untuk pembuatan aplikasi-aplikasi dekstop.
System.Windows.Form	
System.Windows	Namespace yang berhubungan dengan Windows Presentation Foundation (WPF).
System.Windows.Controls	
System.Windows.Shapes	
System.Linq	Namespace yang diperlukan untuk pemrograman LINQ API.
System.Xml.Linq	
System.Data.Linq	
System.Web	Namespace yang diperlukan saat membuat aplikasi-aplikasi Web menggunakan ASP.NET.
System.Security	Namespace yang diperlukan untuk membuat aplikasi-aplikasi yang menekankan keamanan sistem, kriptografi, izin-izin, dan sebagainya.
System.Xml	Namespace yang diperlukan saat aplikasi bahasa C# berinteraksi dengan berkas XML.
System.Threading	Namespace yang diperlukan saat kita mengembangkan aplikasi-aplikasi multithreading.
System.InteropServices	Namespace yang diperlukan saat kita mengembangkan .NET yang perlu mengakses aplikasi yang dikembangkan menggunakan bahasa-bahasa lain, misalnya DLL yang dikembangkan menggunakan bahasa C.

2. Class

Class adalah sebuah deskripsi/gambaran dari sekumpulan objek yang memiliki operasi, attribute dan hubungan antar objek yang sama. Sebuah Class sejatinya adalah cetak biru (blueprint) dari sebuah obyek. Maksudnya, sebuah Class nantinya akan menentukan struktur yang akan digunakan oleh sebuah obyek ketika kita membuat Instance dari Class tersebut.

3. Method

Sederhananya, sebuah method adalah blok kode program yang terdiri dari serangkaian instruksi. Dengan memanggil sebuah method, instruksi-instruksi di dalamnya akan dieksekusi.

Dalam bahasa C#, method Main() adalah titik masuk atau entry point dari aplikasi yang kita buat. Maksudnya, ketika kita menjalankan aplikasi tersebut, Main() adalah method yang pertama kali dipanggil.

4. Variabel

Variabel adalah suatu tempat untuk menyimpan data berupa nilai atau referensi ke sebuah obyek.

2.2 Variabel

Suatu variabel hanyalah suatu penampung data atau nilai di dalam program. Di dalam dunia pemrograman, setiap variabel memiliki tipe data dan dalam kebanyakan bahasa pemrograman, termasuk C#, tipe data suatu variabel ditentukan ketika variabel tersebut dinyatakan atau diisi. Terdapat beberapa cara untuk menyatakan variabel:

- [tipe data] [identifier];
- [tipe data] [identifier] = [nilai];
- [tipe data] [identifier 1], [identifier 2], [identifier N];
- [tipe data] [identifier 1] = [nilai 1], [identifier 2] = [nilai 2], [identifier N] = [nilai N];

Sebagai contoh

```
int jumlahSemuaBarang;
string namaPelanggan = "Fandi";
double harga1, harga2, harga3;
int jumlah1 = 10, jumlah2 = 7, jumlah3 = 12;
```

2.3 Tipe Data

Tipe data adalah jenis-jenis data yang akan disimpan di dalam variabel. Seperti data teks, angka, huruf, dll. Pada C#, ada beberapa jenis tipe data yang umum digunakan.

C# mendukung dua macam tipe data yang digunakan untuk merepresentasikan informasi di dunia nyata. Yaitu, value types atau tipe nilai dan reference type atau tipe referensi.

1) Value Types

   Tipe ini memuat nilai sebenarnya dari data yang tersimpan. Maksudnya, jika Anda memiliki variabel bilangan bulat yang digunakan untuk menyimpan angka 3, angka 3 ini lah yang disimpan di dalam variabel yang Anda deklarasikan. Value type ini tidak dapat dibiarkan kosong atau bernilai null.

2) Reference Types

   Berbeda dengan value type yang menyimpan nilai sebenarnya di dalam variabel yang dideklarasikan, reference type hanya menyimpan alamat yang menunjuk sebuah lokasi di mana nilai atau obyek sebenarnya disimpan. Reference type ini dimungkinkan untuk bernilai null. Maksudnya, tidak ada referensi alamat yang tersimpan di dalam variabel yang Anda deklarasikan.

Tabel berikut mencantumkan tipe data yang tersedia (pre-defined typese) di C# bersama dengan rentang nilai yang dimungkinkan untuk setiap tipe data:

Alias	.NET Type	Deskripsi	Ukuran (bytes)	Rentang Nilai
int	System.Int32	Bilangan bulat	4	-2,147,483,648 ~ 2,147,483,647
long	System.Int64	Bilangan bulat (rentang nilai lebih besar)	8	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
float	System.Single	Bilangan dengan nilai pecahan	4	+/-3.4 x 10^38
double	System.Double	Bilangan dengan nilai pecahan dengan presisi ganda (lebih akurat)	8	+/-1.7 x 10^308
decimal	System.Decimal	Nilai mata uang	16	28 angka penting
char	System.Char	Karakter tunggal	2	Tidak tersedia
bool	System.Boolean	Boolean	1	True atau false
DateTime	System.DateTime	Waktu pada tanggal	8	0:00:00 pada 01/01/0001 sampai 23:59:59 pada 12/31/9999
string	System.String	Rentetan karakter	2 per karakter	Tidak tersedia

2.4 Jenis-jenis Casting

Casting adalah kondisi dimana kita menetapkan nilai dari satu jenis data ke jenis lainnya (konversi).

Ada 2 jenis casting dalam C# yaitu:

1) Implisit Casting (secara otomatis), mengubah jenis yang lebih kecil menjadi ukuran jenis yang lebih besar. Implisit casting dilakukan secara otomatis saat meneruskan jenis ukuran yang lebih kecil ke jenis ukuran yang lebih besar.

   char-> int-> long-> float->double

```
int myInt = 9;
double myDouble = myInt;       // Automatic casting: int to double


Console.WriteLine(myInt);      // Outputs 9
Console.WriteLine(myDouble);   // Outputs 9
```

2) Explicit Casting (secara manual), mengonversi jenis yang lebih besar ke jenis ukuran yang lebih kecil. Ini harus dilakukan secara manual dengan menempatkan jenis dalam tanda kurung di depan nilai.

   double-> float-> long-> int->char

```
double myDouble = 9.78;
int myInt = (int) myDouble;    // Manual casting: double to int


Console.WriteLine(myDouble);   // Outputs 9.78
Console.WriteLine(myInt);      // Outputs 9
```

2.5 Metode Konversi

Metode ini memungkinkan untuk mengkonversi tipe data secara eksplisit dengan menggunakan **built-in metode**, seperti *Convert.ToBoolean, Convert.ToDouble, Convert.ToString, Convert.ToInt32(int)* dan *Convert.ToInt64(long)*

```
int myInt = 10;
double myDouble = 5.25;
bool myBool = true;

Console.WriteLine(Convert.ToString(myInt));    // convert int to string
Console.WriteLine(Convert.ToDouble(myInt));    // convert int to double
Console.WriteLine(Convert.ToInt32(myDouble));  // convert double to int
Console.WriteLine(Convert.ToString(myBool));   // convert bool to string
```

Meskipun dalam mengubah tipe data tidak diperlukan metode konversi nampun pada kasus tertentu metode konversi harus digunakan.

2.6 Operator Aritmatika

Operator aritmatika digunakan untuk melakukan operasi matematika umum:

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	x++
--	Decrement	Decreases the value of a variable by 1	x--

2.7 Operator Penugasan

Operator penugasan digunakan untuk menetapkan nilai ke variabel.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

2.8 Operator Perbandingan

Operator perbandingan digunakan untuk membandingkan dua nilai:

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

2.9 Operator Logika

Operator logika digunakan untuk menentukan logika antara variabel atau nilai:

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

2.10 Operator Matematika

Kelas C# Math memiliki banyak metode yang memungkinkan Anda melakukan tugas matematika pada angka.

- *Math.Max(x,y)*

  Metode ini digunakan untuk mencari nilai tertinggi

```
Math.Max(5, 10);
```

- *Math.Min(x,y)*

  Metode ini dapat digunakan untuk mencari nilai terendah dari x dan y

```
Math.Min(5, 10);
```

- *Math.Sqrt(x)*

  Metode ini mengembalikan akar kuadrat dari x

```
Math.Sqrt(64);
```

- *Math.Abs(x)*

  Metode ini mengembalikan (positif) nilai absolut dari *x*

```
Math.Abs(-4.7);
```

- *Math.Round()*

  Metode ini digunakan untuk membulatkan angka ke bilangan bulat terdekat

```
Math.Round(9.99);
```

BAB III

PERCABANGAN DAN PERULANGAN

3.1 If… Else If… Else

If digunakan jika pernyataan yang akan di eksekusi bernilai true

```
if (condition)
{
  // block of code to be executed if the condition is True
}
```

Else If digunakan jika pernyataan tersebut dieksekusi akan menentukan kondisi baru jika kondisinya false

```
if (condition1)
{
  // block of code to be executed if condition1 is True
}
else if (condition2)
{
  // block of code to be executed if the condition1 is false and condition2
is True
}
else
{
  // block of code to be executed if the condition1 is false and condition2
is False
}
```

Else digunakan jika kondisi yang akan dieksekusi bernilai false

```
if (condition)
{
  // block of code to be executed if the condition is True
}
else
{
  // block of code to be executed if the condition is False
}
```

3.2 Switch Case

Switch digunakan untuk memilih salah satu dari banyak blok kode yang akan dieksekusi.

```
switch(expression)
{
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
    break;
}
```

3.3 Perulangan While

Perulangan while akan selalu dilakukan selama kondisi yang ditentukan True.

```
while (condition)
{
  // code block to be executed
}
```

Contoh

```
int i = 0;
while (i < 5)
{
  Console.WriteLine(i);
  i++;
}
```

Catatan: Jangan lupa untuk menambah variabel yang digunakan dalam kondisi tersebut, jika tidak loop tidak akan pernah berakhir!

3.4 Perulangan Do / While

do/while adalah varian dari while. Perulangan ini akan mengeksekusi blok kode satu kali, sebelum memeriksa apakah kondisinya benar, maka akan mengulang selama kondisinya benar.

```
do
{
  // code block to be executed
}
while (condition);
```

Contoh

```
int i = 0;
do

{

  Console.WriteLine(i);

  i++;

}

while (i < 5);
```

3.5 Perulangan For

Perulangan ini digunakan saat kita tahu persis berapa kali kita ingin mengulang melalui blok kode.

```
for (statement 1; statement 2; statement 3)

{

  // code block to be executed

}
```

Pernyataan 1 dijalankan (satu kali) sebelum eksekusi blok kode.

Pernyataan 2 mendefinisikan kondisi untuk mengeksekusi blok kode.

Pernyataan 3 dieksekusi (setiap kali) setelah blok kode dijalankan.

Contoh

```
for (int i = 0; i < 5; i++)

{

  Console.WriteLine(i);

}
```

3.6 Perulangan Foreach

Foreach digunakan secara eksklusif untuk melakukan perulangan melalui elemen dalam sebuah **array**.

```
foreach (type variableName in arrayName)
{
  // code block to be executed
}
```

Contoh

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
foreach (string i in cars)
{
  Console.WriteLine(i);
}

//output :
Volvo
BMW
Ford
Mazda
```

BAB IV

ARRAY

4.1 Membuat Array

Array digunakan untuk menyimpan beberapa nilai dalam satu variabel, alih-alih mendeklarasikan variabel terpisah untuk setiap nilai. Untuk mendeklarasikan sebuah array, tentukan tipe variabel dengan **tanda kurung siku**

```
string[] cars;
```

untuk mengakses elemen array dengan mengacu pada nomor indeks.

Pernyataan ini mengakses nilai elemen pertama di **mobil**:

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
Console.WriteLine(cars[0]);
// Outputs Volvo
```

Catatan: Indeks array dimulai dengan 0: [0] adalah elemen pertama. [1] adalah elemen kedua, dll.

4.2 Panjang Array

Untuk mengetahui berapa banyak elemen yang dimiliki array, gunakan **Length** properti:

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
Console.WriteLine(cars.Length);
// Outputs 4
```

Panjang array juga bisa digunakan sebagai batas perulangan, contohnya seperti berikut:

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.Length; i++)
{
  Console.WriteLine(cars[i]);
}
```

4.3 Mengurutkan Array

1) *Sort()* mengurutkan array menurut abjad atau dalam urutan menaik

```
// Sort a string
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
Array.Sort(cars);
foreach (string i in cars)
{
  Console.WriteLine(i);
}
// Sort an int
int[] myNumbers = {5, 1, 8, 9};
Array.Sort(myNumbers);
foreach (int i in myNumbers)
{
  Console.WriteLine(i);
}
```

2) *Min(), Max(), Sum()*

Min digunakan untuk mencari nilai terkecil, Max untuk mencari nilai terbesar dan Sum untuk menjumlahkan nilai. Perlu dicatat, metode tersebut hanya berfunngsi jika kita menambahkan namespace ***System.Linq***

```
using System;
using System.Linq;

namespace MyApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      int[] myNumbers = {5, 1, 8, 9};
      Console.WriteLine(myNumbers.Max());  // returns the largest value
      Console.WriteLine(myNumbers.Min());  // returns the smallest value
      Console.WriteLine(myNumbers.Sum());  // returns the sum of elements
    }
  }
}
```

4.4 Cara Lain Dalam Membuat Array

Jika Anda terbiasa dengan C#, Anda mungkin pernah melihat array yang dibuat dengan **new** kata kunci, dan mungkin Anda juga pernah melihat array dengan ukuran tertentu. Di C#, ada berbagai cara untuk membuat array:

```
// Create an array of four elements, and add values later
string[] cars = new string[4];

// Create an array of four elements and add values right away
string[] cars = new string[4] {"Volvo", "BMW", "Ford", "Mazda"};

// Create an array of four elements without specifying the size
string[] cars = new string[] {"Volvo", "BMW", "Ford", "Mazda"};
```

```
// Create an array of four elements, omitting the new keyword, and without
specifying the size

string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

Namun, Anda harus mencatat bahwa jika Anda mendeklarasikan sebuah array dan menginisialisasinya nanti, Anda harus menggunakan new kata kunci:

```
// Declare an array

string[] cars;



// Add values, using new

cars = new string[] {"Volvo", "BMW", "Ford"};



// Add values without using new (this will cause an error)

cars = {"Volvo", "BMW", "Ford"};
```

BAB V

METHODE

5.1 Methode

Sebuah metode adalah blok kode yang hanya berjalan ketika dipanggil. Anda dapat meneruskan data, yang dikenal sebagai parameter, ke dalam metode. Metode digunakan untuk melakukan tindakan tertentu, dan juga dikenal sebagai fungsi. Mengapa menggunakan metode? Untuk menggunakan kembali kode: tentukan kode sekali, dan gunakan berkali-kali.

Sebuah metode didefinisikan dengan nama metode, diikuti oleh tanda kurung **()** . C# menyediakan beberapa metode yang telah ditentukan sebelumnya, yang sudah Anda kenal, seperti Main(), tetapi Anda juga dapat membuat metode Anda sendiri untuk melakukan tindakan tertentu:

```
class Program
{
  static void MyMethod()
  {
    // code to be executed
  }
}
```

- **MyMethod()** itu nama metodenya
- **Static** berarti metode tersebut milik kelas Program dan bukan objek dari kelas Program. Anda akan belajar lebih banyak tentang objek dan cara mengakses metode melalui objek nanti dalam tutorial ini.
- **Void** berarti metode ini tidak memiliki nilai balik. Anda akan mempelajari lebih lanjut tentang nilai pengembalian nanti di bab ini

Untuk memanggil (mengeksekusi) suatu metode, tulis nama metode diikuti oleh dua tanda kurung **()** dan titik koma **;**

```
static void MyMethod()
{
  Console.WriteLine("I just got executed!");
}

static void Main(string[] args)
{
  MyMethod();
}

// Outputs "I just got executed!"
```

5.2 Parameter dalam Method

1) Parameter dan Argumen

Informasi dapat diteruskan ke metode sebagai parameter. Parameter bertindak sebagai variabel di dalam metode. Mereka ditentukan setelah nama metode, di dalam tanda kurung. Anda dapat menambahkan parameter sebanyak yang Anda inginkan, cukup pisahkan dengan koma. Contoh berikut memiliki metode yang menggunakan string nama f yang dipanggil sebagai parameter. Saat metode dipanggil, kami memberikan nama depan, yang digunakan di dalam metode untuk mencetak nama lengkap:

```
static void MyMethod(string fname)
{
  Console.WriteLine(fname + " Refsnes");
}

static void Main(string[] args)
```

```
{
  MyMethod("Liam");
  MyMethod("Jenny");
  MyMethod("Anja");
}

// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes
```

2) Nilai Parameter Default

Anda juga dapat menggunakan nilai parameter default, dengan menggunakan tanda sama dengan (=). Jika kita memanggil metode tanpa argumen, metode ini menggunakan nilai default ("Norwegia"):

```
static void MyMethod(string country = "Norway")
{
  Console.WriteLine(country);
}

static void Main(string[] args)
{
  MyMethod("Sweden");
  MyMethod("India");
  MyMethod();
  MyMethod("USA");
}
// Sweden
// India
// Norway
// USA
```

3) Parameter banyak

Anda dapat memiliki parameter sebanyak

```
static void MyMethod(string fname, int age)
{
  Console.WriteLine(fname + " is " + age);
}

static void Main(string[] args)
{
  MyMethod("Liam", 5);
  MyMethod("Jenny", 8);
  MyMethod("Anja", 31);
}

// Liam is 5
// Jenny is 8
// Anja is 31
```

4) Mengembalikan Nilai

Kata void kunci, yang digunakan dalam contoh di atas, menunjukkan bahwa metode tidak boleh mengembalikan nilai. Jika Anda ingin metode mengembalikan nilai, Anda dapat menggunakan tipe data primitif (seperti int or double) alih-alih void, dan menggunakan return kata kunci di dalam metode:

```
static int MyMethod(int x)
{
  return 5 + x;
}

static void Main(string[] args)
```

```
{
  Console.WriteLine(MyMethod(3));
}

// Outputs 8 (5 + 3)
```

Contoh ini mengembalikan jumlah dari **dua parameter** metode :

```
static int MyMethod(int x, int y)
{
  return x + y;
}

static void Main(string[] args)
{
  Console.WriteLine(MyMethod(5, 3));
}

// Outputs 8 (5 + 3)
```

5) Argumen Bernama

Dimungkinkan juga untuk mengirim argumen dengan **key: values** intaks. Dengan begitu, urutan argumen tidak menjadi masalah:

```
static void MyMethod(string child1, string child2, string child3)
{
  Console.WriteLine("The youngest child is: " + child3);
}

static void Main(string[] args)
{
  MyMethod(child3: "John", child1: "Liam", child2: "Liam");
```

```
}


// The youngest child is: John
```

Argumen bernama sangat berguna saat Anda memiliki beberapa parameter dengan nilai default, dan Anda hanya ingin menentukan salah satunya saat Anda memanggilnya:

```
static void MyMethod(string child1 = "Liam", string child2 = "Jenny", string
child3 = "John")

{

  Console.WriteLine(child3);

}



static void Main(string[] args)

{

  MyMethod("child3");

}



// John
```

5.3 Methode Overloading

```
int MyMethod(int x)

float MyMethod(float x)

double MyMethod(double x, double y)
```

Perhatikan contoh berikut, yang memiliki dua metode yang menambahkan nomor dari jenis yang berbeda:

```
static int PlusMethodInt(int x, int y)

{

  return x + y;

}
```

```
static double PlusMethodDouble(double x, double y)
{
  return x + y;
}

static void Main(string[] args)
{
  int myNum1 = PlusMethodInt(8, 5);
  double myNum2 = PlusMethodDouble(4.3, 6.26);
  Console.WriteLine("Int: " + myNum1);
  Console.WriteLine("Double: " + myNum2);
}
```

Daripada mendefinisikan dua metode yang seharusnya melakukan hal yang sama, lebih baik membebani satu metode.

Dalam contoh di bawah ini, kami membebani PlusMethod metode agar berfungsi untuk keduanya int dan double:

```
static int PlusMethod(int x, int y)
{
  return x + y;
}

static double PlusMethod(double x, double y)
{
  return x + y;
}

static void Main(string[] args)
{
  int myNum1 = PlusMethod(8, 5);
```

```
  double myNum2 = PlusMethod(4.3, 6.26);

  Console.WriteLine("Int: " + myNum1);

  Console.WriteLine("Double: " + myNum2);

}
```

BAB VI

OOP

6.1 Pengertian OOP

OOP adalah singkatan dari *Object Oriented Programming* atau bisa disebut juga sebagai Pemrograman berorientasi objek. OOP yaitu pemrograman berdasarkan konsep objek yang dapat berisi data dalam bentuk field atau dikenal sebagai atribut, serta kode dalam bentuk fungsi (method).

OOP memiliki keunggulan diantaranya:

- Lebih cepat dan lebih mudah dijalankan
- Membantu menjaga kode C# DRY "Don't Repeat Yourself", dan membuat kode mudah untuk dipelihara, dimodifikasi, dan dijalankan.
- Memungkinkan untuk membuat aplikasi yang dapat digunakan kembali secara penuh dengan kode yang lebih sedikit dan waktu pengembangan yang lebih singkat

6.2 Kelas dan objek

Kelas dan objek adalah dua aspek utama dalam OOP. kelas adalah templat untuk objek, dan objek adalah turunan dari kelas. Ketika objek individu dibuat, mereka mewarisi semua variabel dan metode dari kelas. Dalam kelas terdapat dua anggota kelas yaitu field dan metode.

```
// The class
class MyClass
{
  // Class members
  string color = "red";        // field
  int maxSpeed = 200;          // field
  public void fullThrottle()   // method
  {
    Console.WriteLine("The car is going as fast as it can!");
  }
}
```

6.3 Konstruktor

Konstruktor adalah metode khusus yang digunakan untuk menginisialisasi objek. Keuntungan dari konstruktor, adalah bahwa ia dipanggil ketika sebuah objek kelas dibuat. Ini dapat digunakan untuk mengatur nilai awal untuk bidang:

```
// Create a Car class
class Car
{
  public string model;  // Create a field

  // Create a class constructor for the Car class
  public Car()
  {
    model = "Mustang"; // Set the initial value for model
  }

  static void Main(string[] args)
  {
    Car Ford = new Car();  // Create an object of the Car Class (this will
call the constructor)
    Console.WriteLine(Ford.model);  // Print the value of model
  }
}

// Outputs "Mustang"
```

Konstruktor juga dapat mengambil parameter, yang digunakan untuk menginisialisasi bidang.

```
class Car
{
  public string model;

  // Create a class constructor with a parameter
```

```
  public Car(string modelName)
  {
    model = modelName;
  }

  static void Main(string[] args)
  {
    Car Ford = new Car("Mustang");
    Console.WriteLine(Ford.model);
  }
}

// Outputs "Mustang"
```

6.4 Akses Modifier

Akses Modifier yaitu pengubah hak akses yang digunakan untuk mengatur tingkat akses / visibilitas untuk kelas, field, metode, dan properti.

Modifier	Description
`public`	The code is accessible for all classes
`private`	The code is only accessible within the same class
`protected`	The code is accessible within the same class, or in a class that is inherited from that class. You will learn more about inheritance in a later chapter
`internal`	The code is only accessible within its own assembly, but not from another assembly. You will learn more about this in a later chapter

6.5 Properti C#

Sebelum kami mulai menjelaskan properti, Anda harus memiliki pemahaman dasar tentang "Enkapsulasi". Arti dari Enkapsulasi, adalah untuk memastikan bahwa data "sensitif" disembunyikan dari pengguna. Untuk melakukan ini kita harus:

- mendeklarasikan field / variabel sebagai private
- menyediakan public get dan setmetode, melalui properti , untuk mengakses dan memperbarui nilai private bidang

Sebelumnya kita telah belajar bahwa private variabel hanya dapat diakses di dalam kelas yang sama (kelas luar tidak memiliki akses). Namun, terkadang kita perlu mengaksesnya - dan itu bisa dilakukan dengan properti.

Properti itu seperti kombinasi variabel dan metode, dan memiliki dua metode: *get* dan *set* metode:

```
class Person
{
  private string name; // field
  public string Name   // property
  {
    get { return name; }
    set { name = value; }
  }
}

class Program
{
  static void Main(string[] args)
  {
    Person myObj = new Person();
    myObj.Name = "Liam";
    Console.WriteLine(myObj.Name);
  }
}

//Output: Liam
```

6.6 Properti Short-Hand

C# juga menyediakan cara untuk menggunakan properti short-hand / otomatis, di mana kita tidak perlu menentukan field untuk properti, dan kita hanya perlu menulis get; dan set; di dalam properti.

Contoh berikut akan memberikan hasil yang sama seperti contoh di atas. Satu-satunya perbedaan adalah ada lebih sedikit kode:

```
class Person
{
  public string Name  // property
  { get; set; }
}

class Program
{
  static void Main(string[] args)
  {
    Person myObj = new Person();
    myObj.Name = "Liam";
    Console.WriteLine(myObj.Name);
  }
}

//Output: Liam
```

6.7 Inheritance

Dalam C#, dimungkinkan untuk mewarisi field dan metode dari satu kelas ke kelas lainnya. Kami mengelompokkan "konsep inheritance" menjadi dua kategori:

- Derived Class (anak) - kelas yang diturunkan dari kelas lain
- Base Class (induk) - kelas yang diwarisi

Untuk mewarisi dari kelas, gunakan : simbol.

Dalam contoh di bawah ini, *Car* kelas (anak) mewarisi field dan metode dari *Vehicle* kelas (induk):

Contoh:

```
class Vehicle  // base class (parent)
{
  public string brand = "Ford";  // Vehicle field
  public void honk()             // Vehicle method
```

```
  {
    Console.WriteLine("Tuut, tuut!");
  }
}

class Car : Vehicle  // derived class (child)
{
  public string modelName = "Mustang";  // Car field
}

class Program
{
  static void Main(string[] args)
  {
    // Create a myCar object
    Car myCar = new Car();

    // Call the honk() method (From the Vehicle class) on the myCar object
    myCar.honk();

    // Display the value of the brand field (from the Vehicle class) and the
value of the modelName from the Car class
    Console.WriteLine(myCar.brand + " " + myCar.modelName);
  }
}
```

6.8 Polymorphism

Polymorphism berarti "banyak bentuk", dan itu terjadi ketika kita memiliki banyak kelas yang terkait satu sama lain melalui warisan.

Seperti yang sudah dibahas sebelumnya; Pewarisan memungkinkan kita mewarisi bidang dan metode dari kelas lain. Polymorphism menggunakan metode tersebut untuk melakukan tugas yang berbeda. Ini memungkinkan untuk melakukan satu tindakan dengan cara yang berbeda.

Misalnya, dalam base kelas yang disebut *Animal* yang memiliki metode yang disebut *animalSound()*. Kelas Hewan yang diturunkan dapat berupa Babi, Kucing, Anjing, Burung - Dan mereka juga memiliki penerapan suara hewan sendiri (babi oinks, kucing mengeong, dll.):

Contoh:

```
class Animal  // Base class (parent)
{
  public void animalSound()
  {
    Console.WriteLine("The animal makes a sound");
  }
}

class Pig : Animal  // Derived class (child)
{
  public void animalSound()
  {
    Console.WriteLine("The pig says: wee wee");
  }
}

class Dog : Animal  // Derived class (child)
{
  public void animalSound()
  {
    Console.WriteLine("The dog says: bow wow");
  }
}
```

Sekarang kita dapat membuat *Pig* dan *Dog* objek dan memanggil *animalSound()* metode pada keduanya:

```
class Animal  // Base class (parent)
{
  public void animalSound()
  {
    Console.WriteLine("The animal makes a sound");
  }
}

class Pig : Animal  // Derived class (child)
{
  public void animalSound()
  {
    Console.WriteLine("The pig says: wee wee");
  }
}

class Dog : Animal  // Derived class (child)
{
  public void animalSound()
  {
    Console.WriteLine("The dog says: bow wow");
  }
}

class Program
{
  static void Main(string[] args)
  {
    Animal myAnimal = new Animal();  // Create a Animal object
    Animal myPig = new Pig();  // Create a Pig object
```

```
    Animal myDog = new Dog();  // Create a Dog object


    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
  }
}
```

Output:

The animal makes a sound

The animal makes a sound

The animal makes a sound

Output dari contoh di atas mungkin tidak seperti yang Kita harapkan. Itu karena metode kelas dasar (base class) mengganti metode kelas turunan, ketika mereka berbagi nama yang sama. Namun, C# menyediakan opsi untuk mengganti metode kelas dasar, dengan menambahkan *virtual* ke metode di dalam kelas dasar, dan dengan menggunakan *override* kata kunci untuk setiap metode kelas turunan:

```
class Animal  // Base class (parent)
{
  public virtual void animalSound()
  {
    Console.WriteLine("The animal makes a sound");
  }
}


class Pig : Animal  // Derived class (child)
{
  public override void animalSound()
  {
    Console.WriteLine("The pig says: wee wee");
  }
```

```
}


class Dog : Animal  // Derived class (child)
{
  public override void animalSound()
  {
    Console.WriteLine("The dog says: bow wow");
  }
}


class Program
{
  static void Main(string[] args)
  {
    Animal myAnimal = new Animal();  // Create a Animal object
    Animal myPig = new Pig();  // Create a Pig object
    Animal myDog = new Dog();  // Create a Dog object

    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
  }
}
```

Output:

The animal makes a sound

The pig says: wee wee

The dog says: bow wow

6.9 Enum

Enum adalah "kelas" khusus yang mewakili sekelompok konstanta (variabel tidak dapat diubah / hanya-baca). Enum adalah kependekan dari "enumerations", yang berarti "terdaftar secara khusus".

Contoh:

```
enum Level
{
  Low,
  Medium,
  High
}
```

Cara mengaksesnya:

```
Level myVar = Level.Medium;
Console.WriteLine(myVar);
```

Kita juga dapat memiliki enum di dalam kelas:

Contoh:

```
class Program
{
  enum Level
  {
    Low,
    Medium,
    High
  }
  static void Main(string[] args)
  {
    Level myVar = Level.Medium;
    Console.WriteLine(myVar);
  }}
```

Nilai Enum secara default item pertama enum memiliki nilai 0. Item kedua memiliki nilai 1, dan seterusnya. Untuk mendapatkan nilai integer dari sebuah item, Kita harus secara eksplisit mengonversi item tersebut menjadi int:

Contoh:

```
enum Months
{
  January,    // 0
  February,   // 1
  March,      // 2
  April,      // 3
  May,        // 4
  June,       // 5
  July        // 6
}

static void Main(string[] args)
{
  int myNum = (int) Months.April;
  Console.WriteLine(myNum);
}
```

Output:

3

Kita juga dapat menetapkan nilai enum kita sendiri, dan item berikutnya akan memperbarui angkanya sesuai:

Contoh:

```
enum Months
{
  January,    // 0
  February,   // 1
```

```
  March=6,    // 6
  April,      // 7
  May,        // 8
  June,       // 9
  July        // 10
}

static void Main(string[] args)
{
  int myNum = (int) Months.April;
  Console.WriteLine(myNum);
}
```

Output:

7

Selain itu, Enum sering digunakan dalam pernyataan *switch* untuk memeriksa nilai yang sesuai:

Contoh:

```
enum Level
{
  Low,
  Medium,
  High
}

static void Main(string[] args)
{
  Level myVar = Level.Medium;
  switch(myVar)
  {
    case Level.Low:
```

```
      Console.WriteLine("Low level");
      break;
    case Level.Medium:
       Console.WriteLine("Medium level");
      break;
    case Level.High:
      Console.WriteLine("High level");
      break;
  }
}
```

Output:

Medium Level

BAB VII

EXCEPTIONS

7.1 Exception

Saat menjalankan kode C#, kesalahan yang berbeda dapat terjadi: kesalahan pengkodean yang dibuat oleh programmer, kesalahan karena input yang salah, atau hal-hal tak terduga lainnya. Ketika terjadi kesalahan, C# biasanya akan berhenti dan menghasilkan pesan kesalahan. Istilah teknis untuk ini adalah: C# akan membuat pengecualian (membuat kesalahan).

7.2 Try and catch

Try pernyataan memungkinkan kita untuk menentukan blok kode yang akan diuji untuk kesalahan ketika sedang dijalankan. Catch pernyataan memungkinkan kita untuk menentukan blok kode yang akan dieksekusi, jika terjadi kesalahan dalam blok try.

```
try
{
  //  Block of code to try
}
catch (Exception e)
{
  //  Block of code to handle errors
}
```

Perhatikan contoh berikut, di mana kita membuat array tiga bilangan bulat:
Ini akan menghasilkan kesalahan, karena myNumbers [10] tidak ada.

```
int[] myNumbers = {1, 2, 3};

Console.WriteLine(myNumbers[10]); // error!
```

Pesan kesalahannya akan seperti ini:

```
System.IndexOutOfRangeException: 'Index was outside the
bounds of the array.'
```

Jika terjadi kesalahan, kita dapat menggunakan try...catch untuk menangkap kesalahan tersebut dan menjalankan beberapa kode untuk menanganinya. Dalam contoh berikut, kami menggunakan variabel di dalam catch block (e) bersama dengan properti built-in Message, yang mengeluarkan pesan yang menjelaskan pengecualian:

```
try
{
  int[] myNumbers = {1, 2, 3};
  Console.WriteLine(myNumbers[10]);
}
catch (Exception e)
{
  Console.WriteLine(e.Message);
}
```

Outputnya:

```
Index was outside the bounds of the array.
```

Kita juga dapat menampilkan pesan kesalahan kita sendiri:

```
try
{
  int[] myNumbers = {1, 2, 3};
  Console.WriteLine(myNumbers[10]);
}
catch (Exception e)
{
  Console.WriteLine("Something went wrong.");
}
```

Outputnya:

```
Something went wrong.
```

7.3 Finally

Finally Pernyataan memungkinkan kita mengeksekusi kode, setelah *try...catch*, terlepas dari hasilnya:

Contoh:

```
try
{
  int[] myNumbers = {1, 2, 3};
  Console.WriteLine(myNumbers[10]);
}
catch (Exception e)
{
  Console.WriteLine("Something went wrong.");
}
finally
{
  Console.WriteLine("The 'try catch' is finished.");
}
```

Outputnya:

```
Something went wrong.
The 'try catch' is finished.
```

7.4 throw keyword

Pernyataan throw memungkinkan kita untuk membuat kesalahan kustom. Pernyataan throw digunakan bersama-sama dengan kelas pengecualian.

Ada banyak kelas pengecualian tersedia di C#: ArithmeticException, FileNotFoundException, IndexOutOfRangeException, TimeOutException, dll:

```
static void checkAge(int age)
{
  if (age < 18)
  {
```

```
    throw new ArithmeticException("Access denied - You must be at least 18
years old.");
  }
  else
  {
    Console.WriteLine("Access granted - You are old enough!");
  }
}

static void Main(string[] args)
{
  checkAge(15);
}
```

Pesan kesalahan yang ditampilkan dalam program ini adalah:

```
System.ArithmeticException: 'Access denied - You must be at least 18
years old.'
```

Jika *age* berusia 20 tahun, Anda tidak akan mendapatkan pengecualian:

```
checkAge(20);
```

Outputnya adalah:

```
Access granted - You are old enough!
```

BAB VIII

KONEKSI DATABASE

8.1 Konsep MVC

Model-View-Controller atau MVC adalah sebuah metode untuk membuat sebuah aplikasi dengan memisahkan data dari tampilan dan cara bagaimana memprosesnya. Dalam implementasinya kebanyakan kerangka kerja dalam aplikasi web adalah berbasis arsitektur MVC.

- *Model*

  Model adalah bagian kode program yang menangani query atau database. Jadi isi dari model merupakan bagian (fungsi-fungsi) yang berhubungan langsung dengan database untuk memanipulasi data seperti memasukkan data, pembaruan data, hapus data, dan lain-lain, namun tidak dapat berhubungan langsung dengan bagian view.

- *View*

  View adalah bagian kode prgram atau pun komponen yang mengatur tampilan

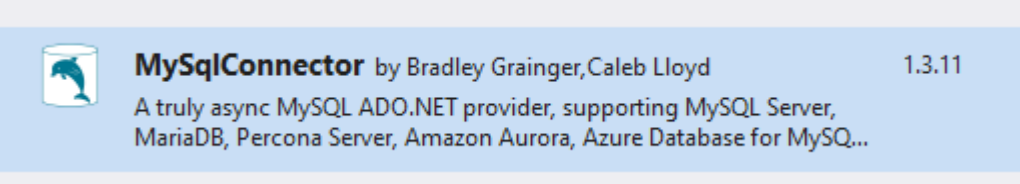
- *Controller*

  Komponen yang digunakan untuk mengontrol semua alur yang diharapkan dari sebuah program. Oleh karena itu, semua logika-logika yang diperlukan berada di komponen Controller.

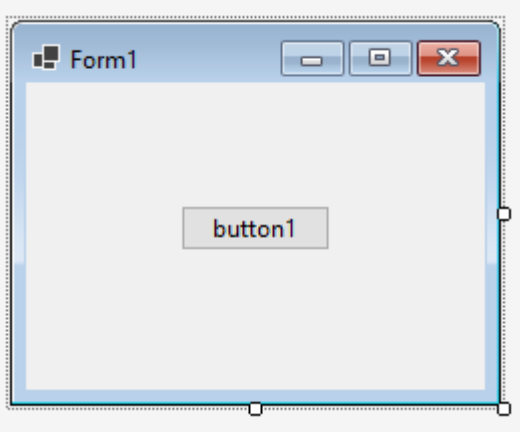
8.2 Membuat Koneksi

Membuat koneksi database pada C# sangat perlu dilakukan untuk untuk mengambil data dari database. Aplikasi yang dibangun dengan bahasa C# sangat identik dengan database karena aplikasi C# digunakan untuk CRUD (Create, Read, Update, Delete).

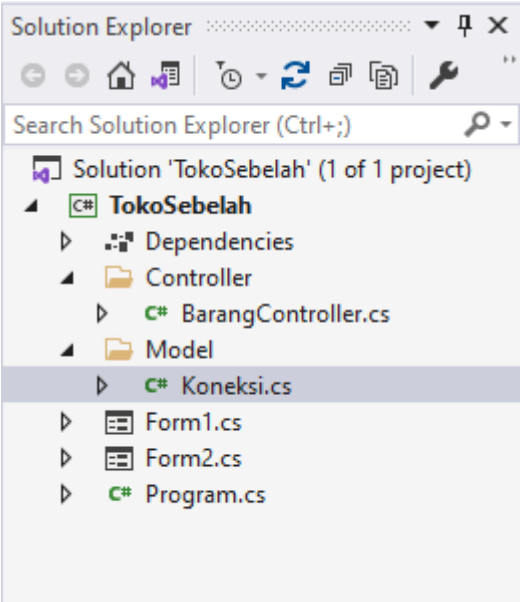
1. Instal NuGet Package mysqlconnector



2. Buat form berisi button



3. Buat Class dengan nama koneksi



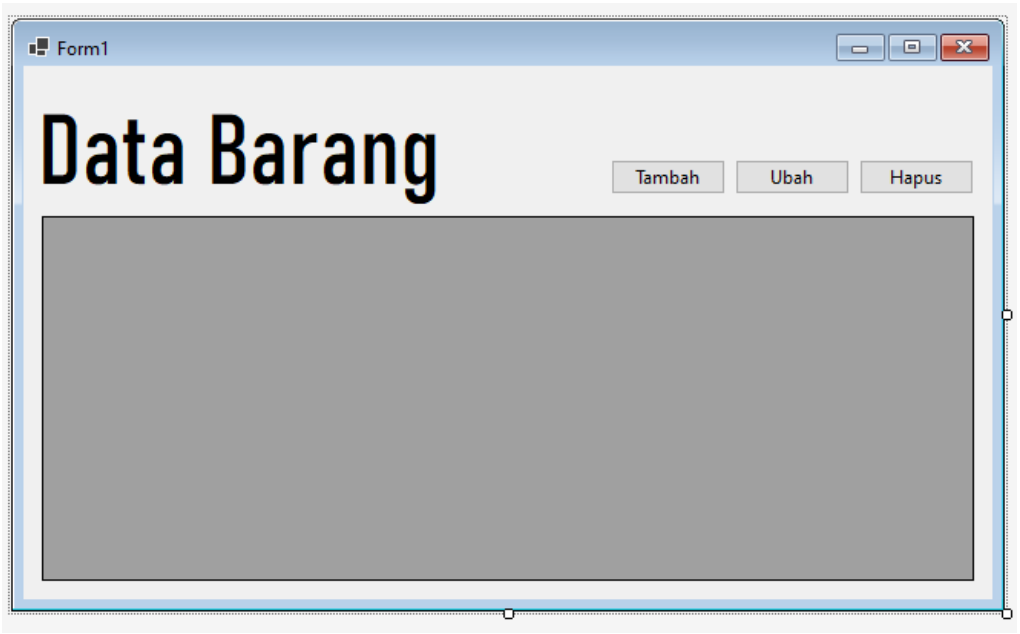
4. Buat program dalam class koneksi seperti berikut

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MySqlConnector;

namespace TokoSebelah.Model
{
    1 reference
    public class Koneksi
    {
        public MySqlCommand cmd;
        public DataSet ds;
        public MySqlDataAdapter da;

        2 references
        public MySqlConnection GetConn()
        {
            MySqlConnection Conn = new MySqlConnection();
            Conn.ConnectionString = "server=localhost; user=root; database=db_contoh";
            try
            {
                Conn.Open();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Koneksi gagal " + ex.Message);
            }
            return Conn;
        }
    }
}
```

8.3 CRUD (CREATE, READ, UPDATE, DELETE)

1. READ

   Yang pertama kita pelajari bagaimana cara menampilkan data dari database kedalam program. Caranya, ikuti Langkah sebelumnya sampai tahap 6. Kemudian sediakan form seperti berikut:



Kemudian buat class dengan nama BararangController, lalu buat fungsi tampil barang

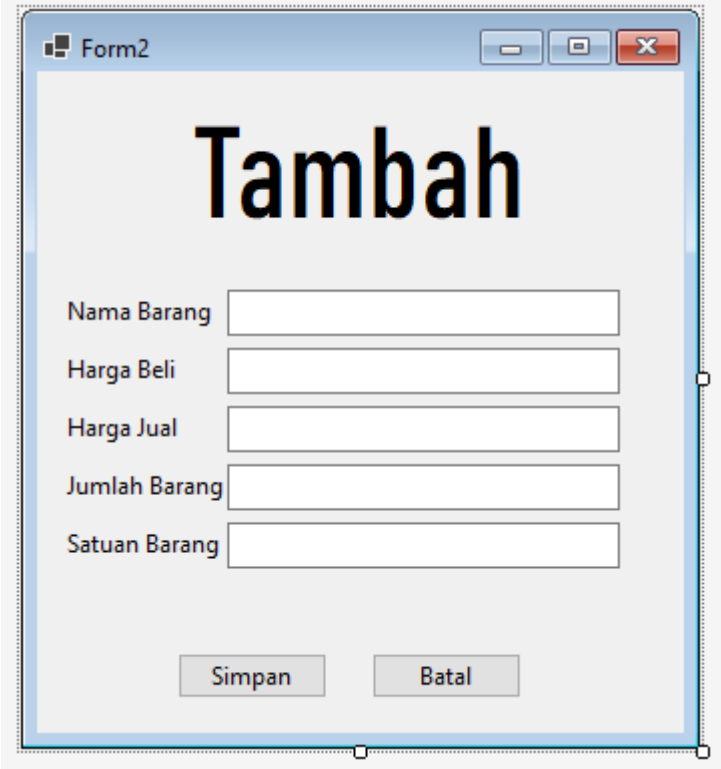
```
1   using System;
2   using System.Collections.Generic;
3   using System.Data;
4   using System.Linq;
5   using System.Text;
6   using System.Threading.Tasks;
7   using MySqlConnector;
8   using System.Windows.Forms;
9
10  namespace TokoSebelah.Controller
11  {
        4 references
12      public class BarangController : Model.Koneksi
13      {
            1 reference
14          public DataTable tampilBarang()
15          {
16              DataTable data = new DataTable();
17              try
18              {
19                  string tampil = "select * from tb_barang";
20                  da = new MySqlDataAdapter(tampil, GetConn());
21                  da.Fill(data);
22              }
23              catch (Exception ex)
24              {
25                  MessageBox.Show(ex.Message);
26              }
27              return data;
28          }
```

Kemudian panggil fungsi tersebut pada form dan masukan datanya ke dalam datagridview

```
using MySqlConnector;
using TokoSebelah.Controller;

namespace TokoSebelah
{
    5 references
    public partial class Form1 : Form
    {
        private BarangController brgController;

        2 references
        public Form1()
        {
            brgController = new BarangController();
            InitializeComponent();
        }

        1 reference
        private void Form1_Load(object sender, EventArgs e)
        {
            Segarkan();

        }

        1 reference
        private void Segarkan()
        {
            dataGridView1.DataSource = brgController.tampilBarang();
            dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
        }
```

2. CREATE

   Cara menambahkan data ke dalam database seperti berikut. Buat form baru seperti berikut:



Kemudian pada class BarangController, buat fungsi/void dengan nama tabah barang

```
1 reference
public void tambahBarang(
    string namabarang, string hargabeli, string hargajual, string jumlahbarang, string satuan)
{
    string tambah = "insert into tb_barang values(" +
        "@KodeBarang,@NamaBarang,@HargaBeli,@HargaJual,@JumlahBarang,@Satuan)";
    try
    {
        cmd = new MySqlCommand(tambah, GetConn());
        cmd.Parameters.Add("@KodeBarang", MySqlDbType.VarChar).Value = "";
        cmd.Parameters.Add("@NamaBarang", MySqlDbType.VarChar).Value = namabarang;
        cmd.Parameters.Add("@HargaBeli", MySqlDbType.VarChar).Value = hargabeli;
        cmd.Parameters.Add("@HargaJual", MySqlDbType.VarChar).Value = hargajual;
        cmd.Parameters.Add("@JumlahBarang", MySqlDbType.VarChar).Value = jumlahbarang;
        cmd.Parameters.Add("@Satuan", MySqlDbType.VarChar).Value = satuan;
        cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Tambah Data Gagal " + ex.Message);
    }
}
```

Kemudian pada panggil fungsi tambahBarang

```
private void button1_Click(object sender, EventArgs e)
{
    brgController = new BarangController();
    brgController.tambahBarang(
        textBox1.Text, textBox2.Text, textBox3.Text, textBox4.Text, textBox5.Text);
    reload();
    textBox1.Focus();
}
```

3. UPDATE

```
1 reference
public void updateBarang(
    string kodebarang, string namabarang, string hargabeli,
    string hargajual, string jumlahbarang, string satuanbarang)
{
    string update = "update tb_barang set " +
        "NamaBarang=@NamaBarang,HargaBeli=@HargaBeli,HargaJual=@HargaJual," +
        "JumlahBarang=@JumlahBarang,SatuanBarang=@SatuanBarang " +
        "where KodeBarang="+kodebarang+"";
    try
    {
        cmd = new MySqlCommand(update, GetConn());
        cmd.Parameters.Add("@KodeBarang", MySqlDbType.VarChar).Value = kodebarang;
        cmd.Parameters.Add("@NamaBarang", MySqlDbType.VarChar).Value = namabarang;
        cmd.Parameters.Add("@HargaBeli", MySqlDbType.VarChar).Value = hargabeli;
        cmd.Parameters.Add("@HargaJual", MySqlDbType.VarChar).Value = hargajual;
        cmd.Parameters.Add("@JumlahBarang", MySqlDbType.VarChar).Value = jumlahbarang;
        cmd.Parameters.Add("@SatuanBarang", MySqlDbType.VarChar).Value = satuanbarang;
        cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Update Data Gagal " + ex.Message);
    }
}
```

4. DELETE

```
1 reference
public void hapusBarang(string kodebarang)
{
    string delete = "delete from tb_barang where kodebarang = @kodebarang";
    try
    {
        cmd = new MySqlCommand(delete, GetConn());
        cmd.Parameters.Add("@kodebarang", MySqlDbType.VarChar).Value = kodebarang;
        cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Hapus Data Gagal " + ex.Message);
    }
}
```