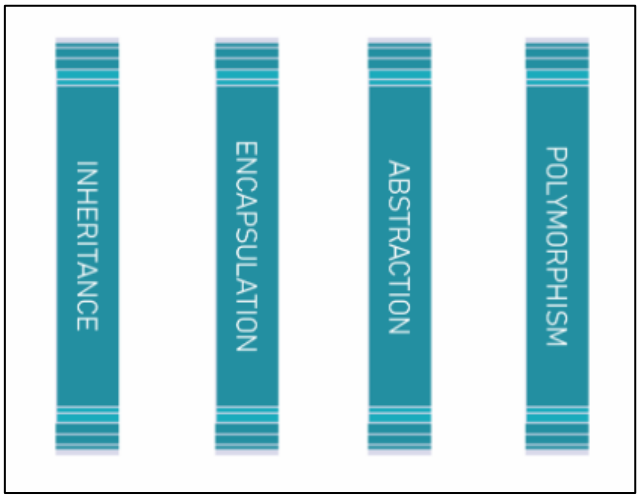
PENGENALAN OOP

Prinsip-Prinsip OOP

OOP dikenal memiliki empat prinsip atau pilar yang menjadi konsep dasar OOP ketika menggunakannya. Berikut masing-masing penjelasannya.



1. Abstraction

Abstraction atau dalam bahasa Indonesia **Abstraksi**, bukanlah mengenai kode program. Abstraksi adalah **suatu proses dimana seorang *programmer* melihat contoh-contoh di kehidupan nyata dan menerjemahkan konsep - konsep yang didapat ke dalam suatu program**.

Sebagai ***contoh***, seorang *game programmer* mula-mula akan melihat suatu mobil memiliki kecepatan, jumlah bensin dan jenis bannya. Sedangkan *programmer retailer otomotif* mungkin akan lebih peduli dengan harga, warna dan mereknya.

Tentu saja mereka dapat memasukkan semua hal tersebut dan hal lainnya ke dalam program. Akan tetapi hal ini sangat bergantung dengan apa yang dilihat oleh programmer.

2. Encapsulation

Encapsulation pada *OOP* adalah **konsep tentang pengikatan data atau metode berbeda yang disatukan atau "dikapsulkan" menjadi satu unit data**. *Encapsulation* dapat memudahkan dalam pembacaan kode karena informasi yang disajikan tidak perlu dibaca secara rinci dan sudah merupakan satu kesatuan.

Sebagai *contoh*, jika kita lihat obat kapsul yang kita makan ketika kita sakit. Di dalamnya terdapat banyak jenis obat dan vitamin yang membuat kita lebih sehat. Ada dua hal yang terkait dengan fakta ini:

- Kita tidak tahu apa yang ada di dalam kapsul. Kita hanya tahu manfaat dan cara kerja kapsul tersebut (ditelan).

- Memakan satu kapsul berarti memakan semua obat dan vitamin di dalamnya sekaligus.

Dengan cara yang sama, suatu class mengkapsulkan semua hal di dalamnya:

- Pengguna class tidak perlu tahu bagian dalam dari class, mereka cukup tahu kegunaan class tersebut.
- Membuat suatu objek class tertentu berarti objek tersebut memiliki semua hal yang ada pada class tersebut. Seperti properties, method, enum dan event.

3. Inheritance

Prinsip *inheritance* pada OOP adalah di mana **kita dapat membentuk class baru yang "mewarisi" atau memiliki bagian-bagian dari class yang sudah ada sebelumnya**. Konsep ini menggunakan sistem hirarki atau bertingkat.

4. Polymorphism

Polymorphism merupakan **kemampuan objek, variabel, atau fungsi yang dapat memiliki berbagai bentuk**.

Sebagai contoh, *Polymorphism* dapat digambarkan seperti seorang yang menguasai dan mampu berbicara dalam beberapa bahasa. Di sini, seseorang yang bertindak sebagai objek dan kemampuan dia berbicara merupakan sebuah *Polymorphism*. Contoh lainnya adalah *smartphone*. Selain sebagai alat komunikasi, *smartphone* yang bertindak sebagai objek dapat digunakan sebagai kamera, pemutar musik dan radio.

Istilah - Istilah pada OOP

Terdapat beberapa istilah yang perlu kita ketahui di dalam OOP, yaitu :

- ***Class***, merupakan pengelompokan. Misalnya mengelompokkan antara kelas hewan dan kelas tumbuhan.
- **Objek**, merupakan isi dari suatu class yang memiliki ciri-ciri unik. Misalnya burung, kucing, sapi termasuk dalam kelas hewan. Sedangkan mawar, melati termasuk dalam kelas tumbuhan.
- ***Property***, merupakan perlengkapan yang dimiliki oleh sebuah objek. Misalnya kucing memiliki property berat, tinggi, warna bulu, dll. Fungsi *property* yaitu untuk menerangkan objek secara jelas.
- ***Method***, merupakan suatu metode atau aksi yang bisa dilakukan oleh suatu objek. Misalnya kucing dapat berjalan, makan, dan minum. (*method* telah kita pelajari pada materi sebelumnya).

Method

Method (Metode) merupakan **kumpulan dari beberapa pernyataan yang digabungkan menjadi satu yang bertujuan untuk melakukan suatu tugas tertentu**. *Method* juga merupakan sekumpulan pernyataan yang akan dijalankan ketika dipanggil pada suatu program.

Method memiliki banyak kelebihan diantaranya adalah :

- Kode bersifat ***reusable*** yang artinya kode tersebut dapat digunakan kembali.
- **Mudah untuk menguji**
- *Method* dapat **menerima lebih dari satu masukan/argumen yang berbeda**

Untuk mendefiniskan suatu *method* pada *C#* :

```
[Access modifier] [tipe data] [nama method] (parameter1, parameter2, ...) {
    // Statements
}
```

Dari struktur di atas:

- **Access modifier**, untuk saat ini, gunakan saja *access modifier* public.
- **Tipe data** merupakan tipe data yang dihasilkan *method*. Bisa berupa void atau tipe data *C#* lainnya, termasuk enum atau *class custom*.
- **Nama method** berupa suatu *identifier* yang valid.
- **Parameter** sekumpulan variabel pada *method* untuk berkomunikasi dengan program di luar *method*. ***Method*** juga bisa tidak memiliki parameter.
- **Statements**, kumpulan perintah yang akan dijalankan ketika *method* dipanggil.

Suatu *method* harus ditempatkan di dalam suatu *class*. *Method* adalah salah satu bentuk dari member suatu *class*. Suatu *method* dipanggil atau dijalankan dengan menuliskan namanya. Sebagai catatan, *program dengan bahasa pemrograman C#* paling tidak memiliki satu *method***, yaitu *method* Main.

Class dan Object

Class

Kita mendefinisikan class di dalam namespace pada C#. Berikut struktur class sederhana pada C# :

```
[access modifier] class [nama class] {
    //Member-member pada class
}
```

Pengunaan access modifiers pada *C#* akan dijelaskan pada bagian ini. Untuk percobaan saat ini, cukup gunakan public untuk semua *class*.

Sebagai contoh, kita dapat membuat *class* Mobil sebagai berikut:

```
namespace Belajar1 {
  class Program {
    static void Main(string[] args) {

    }
  }

  public class Mobil {
    public double kecepatan;
    public double bensin;
    public double posisi;
    public string nama;

    public void percepat() {
      this.kecepatan += 10;
      this.bensin -= 5;
    }
    public void maju() {
      this.posisi += this.kecepatan;
      this.bensin -= 2;
    }
    public void isiBensin(double bensin) {
      this.bensin += bensin;
    }
  }
}
```

Perhatikan bahwa class Mobil berada di dalam namespace Belajar1 setingkat dengan class Program (dalam hal level kurung kurawalnya). Di sini kita telah melakukan proses abstraksi. Pada class Mobil di atas, kecepatan, bensin, posisi dan nama adalah fields, sedangkan percepat, maju dan isiBensin adalah method.

Objek

Suatu class digunakan untuk membuat objek. Untuk membuat objek, kita perlu menginstansiasi (*instantiate*) suatu class dengan salah satu dari cara berikut:

```
[nama class] [nama objek] = new [nama class]();
```

```
[nama class] [nama objek] = new [nama class]() { field1=nilai1, field2=nilai2, ... };
```

```
[nama class] [nama objek] = new [nama class](param1, param2, ...);
```

Ketika kita membuat suatu objek dari suatu class, kita mendapatkan semua member dari class tersebut. Kita dapat mengkases member class tersebut dengan mengetikkan titik dan nama member yang bersangkutan. Sebagai contoh, kita tuliskan dalam class Program:

```
class Program {
  static void Main(string[] args) {
    Mobil SuatuMobil = new Mobil();
    Mobil MobilSaya = new Mobil() { nama = "Ferrari", kecepatan = 0, bensin = 30000, posisi = 0 };
    MobilSaya.Percepat();
    Console.WriteLine(SuatuMobil.nama);
    Console.WriteLine(SuatuMobil.bensin);

    Console.WriteLine(MobilSaya.nama);
    Console.WriteLine(MobilSaya.bensin);
  }
}
```

Program di atas akan menampilkan:

```
#output
[baris kosong]
0
Ferrari
30000
```

Baris pertama kosong dan baris kedua bernilai 0 karena nilai *default* string dan int. Perlu diingat bahwa tidak baik untuk menggunakan variabel (field nama dan bensin) tanpa menginisialisasinya.

Constructor

Sebelum kita dapat menggunakan suatu **objek** kita perlu mengisi nilai pada *field* atau *property* yang ada di dalamnya. Untuk memberi nilai pada *field* atau *property* suatu objek kita dapat menggunakan **method constructor**. Suatu *constructor* didefinisikan dengan cara :

```
public [nama class] (param1, param2, ...) {
    //Hal-hal yang dilakukan ketika suatu objek diinstansiasi (dibuat)
}
```

Sebagai contoh, tambahkan kode berikut ke dalam class Mobil yang telah kita buat sebelumnya :

```
public Mobil(string nama, double kecepatan, double bensin) {
  this.nama = nama;
  this.kecepatan = kecepatan;
  this.bensin = bensin;
  this.posisi = 0;
}
```

Constructor di atas menerima tiga parameter. Parameter yang diberikan digunakan untuk menginisialisasi masing-masing *field* di dalam objek.

- Perhatikan bahwa **variabel this.nama dengan variabel nama di atas adalah dua variabel yang berbeda**. Keyword this pada C# mewakili objek yang sedang aktif pada class.
- Ketika tidak ada parameter nama, variabel nama akan mewakili field nama pada class. Akan sama saja apabila kita menulis this.name atau name.
- Akan tetapi apabila terdapat parameter nama, nama akan mewakili parameter tersebut, dan untuk mewakili *field* nama, kita hanya dapat menggunakan this.nama.

Untuk menggunakan *constructor* untuk membuat objek, kita dapat menggunakan kode seperti berikut:

```
Mobil MobilAnda = new Mobil("Lamborghini", 300, 50000);
```

Perhatikan bahwa karena *constructor* baru kita, perintah-perintah new Mobil() sebelumnya menjadi *error *karena kita telah mendeklarasikan *constructor* kita. Kita dapat membuat *constructor* lain di dalam class Mobil untuk menghindari *error* tersebut:

```
public Mobil() {
  this.nama = "";
  this.kecepatan = 0;
  this.bensin = 0;
  this.posisi = 0;
}
```

Access Modifier

Access modifier menentukan apakah suatu *class, method, field, property, event* atau hal-hal lainnya dapat diakses di luar *class* induknya.

- public membuat member yang bersangkutan dapat diakses dari mana saja.
- private membuat member yang bersangkutan hanya dapat diakses dari dalam class itu sendiri.
- protected membuat member yang bersangkutan hanya dapat diakses dari class itu sendiri atau class lain yang merupakan turunan (*inherits*) dari class yang bersangkutan.
- internal membuat member yang bersangkutan hanya dapat diakses dari *assembly* yang sama, atau dengan kata lain, aplikasi yang sama.
- protected internal menggabungkan protected dan internal, yang berarti member yang bersangkutan hanya dapat diakses dari class itu sendiri atau dari class lain dari dalam *assembly* yang sama yang merupakan turunan dari class yang bersangkutan.

Perlu diketahui juga bahwa **kata kunci static dan virtual bukanlah suatu *access modifier***.

Field dan Property

Field merupakan variabel sederhana yang dideklarasikan di dalam *class*. Sedangkan **property** adalah gabungan sebuah *field private*, sebuah *method getter* dan sebuah *method setter*.

Field pada C#:

```
public int Jumlah;
```

Property pada C#:

```
private int jumlah;
public int Jumlah {
  get {
    return jumlah;
  }
  set {
    jumlah = value;
  }
}
```

Kelebihan menggunakan *properties* dibanding *field* antara lain :

- Dapat menyembunyikan atau menjaga implementasi kode. Sebagai contoh, kita dapat menolak memasukkan suatu nilai ke dalam suatu *properties* apabila nilai tersebut tidak memenuhi kondisi tertentu.
- Kita dapat mencatat akses pembacaan atau perubahan *properties* dengan cara menambahkan kode pelacak ke dalam *method getter* dan *setter*.
- Dapat diakses dengan *reflection* (akan dibahas pada berikutnya).
- Dapat digunakan untuk *data binding*.

Penggunaan *Property*

Properti merupakan *member* pada *class* yang menyediakan mekanisme fleksibel untuk **membaca** (*read*), **menulis** (*write*), atau **menghitung** nilai pada **private field**. **Properti** dapat digunakan seolah-olah mereka adalah anggota *(member)* dari data **publik**, namun menyertakan metode khusus yang disebut **accessors**.

Accessor properti berisi pernyataan eksekusi yang membantu dalam (**membaca** atau **menghitung**) atau menyetel (**menulis**) **field** yang sesuai. Deklarasi **accessor** dapat mencakup **get accessor**, **set accessor**, atau keduanya.

contoh :

```
class PropertyTest
{
    private string materi;

    public string Materi
    {
        get { return materi; }
        set { materi = value; }
    }
}

class Program {
    static void Main(string[] args) {
        PropertyTest p = new PropertyTest();
        p.Materi = "Bahasa Pemrograman C#";
        Console.WriteLine(p.Materi);
    }
}
```

Class PropertyTest diatas mempunyai properti yang bernama materi yang mempunyai dua *accessor* yaitu get dan set, **set accessor** digunakan untuk menetapkan *value* atau nilai pada variabel materi, dan **get accessor** digunakan untuk mengembalikan nilainya. Setelah properti didefinisikan, kita dapat menggunakannya untuk menetapkan dan membaca *private member*.

Mengapa harus menggunakan *property*? Mengapa tidak membuat variabel class member menjadi ***public*** dan mengaksesnya secara langsung?

Dengan **property** kita memiliki pilihan untuk mengontrol logika dalam mengakses variabel. Misalnya, kita dapat memeriksa apakah nilai lebih dari 50 (lima puluh), sebelum menugaskannya ke variabel :

```
class Person
{
  private int nilai = 0;
  public int Nilai
  {
    get { return nnilai; }
    set {
      if (value > 50 )
        nilai = value;
    }
  }
}
```

Inheritance

Inheritance merupakan salah satu yang paling penting dalam OOP atau *Object Oriented Programming* pada bahasa pemrograman *C#*. Kita dapat mendefinisikan suatu *class* menggunakan atau berdasarkan *class* yang lain.

Class yang propertinya diwariskan oleh *class* yang lain disebut dengan **Base Class**.

Berikut adalah contoh deklarasi dari *inheritance* atau warisan dalam bahasa pemrograman *C#*.

```
class Mobil {
  public int Roda {get; set;}
  public int Tahun {get; set;}
}
```

Sekarang kita akan membuat turunan dengan *class* lain :

```
class Civic : Mobil {
  public Civic() {
    Roda = 4;
  }
  public void Klakson() {
    Console.Write("Biiim...!!!");
  }
}
```

Untuk memakai *base class*, kamu perlu menambahkan titik dua : untuk dapat mengaksesnya. Member public pada class Mobil, akan menjadi member public pada class Civic.

Dengan begitu, kamu dapat mengakses member dari class Civic.

contoh :

```
static void Main(string[] args) {
  Civic c = new Civic();
  Console.WriteLine(c.Roda);
  c.Klakson();
}

#output
4
Biiim...!!!
```

Perlu diingat, bahwa member *inheritance* juga mewariskan *method* pada *base class*.

```
class Mobil{
  public void Klakson() {
    Console.WriteLine("Biiim...!!!");
  }
}
class Civic : Mobil {
  int Roda;
}
static void Main(string[] args) {
  Civic c = new Civic();
  c.Klakson();
}

#output
Biiim...!!!
```

Method Klakson dapat dipanggil, yang mana *method* Klakson di deklarasikan pada *base class*.

Polymorphism

Polymorphism merupakan suatu konsep dalam bahasa pemrograman yang mempunyai arti "**sesuatu yang sama dapat memiliki bentuk yang berbeda beda**". Dapat dikatakan bahwa satu *method* pada *class* dapat mempunyai beberapa **implementasi yang berbeda-beda**. *Polymorphism* sendiri berasal dari bahasa Yunani, yang artinya mempunyai banyak bentuk.

Salah satu manfaat dari *polymorphism* adalah ketika kita membuat **program yang memungkinkan pengguna untuk menggambar berbagai bentuk, setiap bentuk digambar secara berbeda**.

```
class Bentuk {
  public virtual void Gambar() {
    Console.Write("Ini adalah Base Class Bentuk");
  }
}
```

Sekarang kita akan membuat class Bentuk yang berbeda menggunakan *method* gambar mereka sendiri.

```
class Lingkaran : Bentuk{
  public override void Gambar() {
    // Menggambar Lingkaran...
    Console.WriteLine("Menggambar Lingkaran...");
  }
}
class Persegi : Bentuk{
  public override void Gambar() {
    // Menggambar Persegi...
    Console.WriteLine("Menggambar Persegi...");
  }
}
```

Method virtual Gambar, pada *base class* dapat diganti (*override*) pada *class* turunan. Dengan demikian class Lingkaran dan juga class Persegi dapat memiliki method Gambar sendiri.

contoh:

```
static void Main(string[] args) {
  Bentuk l = new Lingkaran();
  l.Gambar();
  //Outputs "Menggambar Lingkaran..."

  Bentuk p = new Persegi();
  p.Gambar();
  //Outputs "Menggambar Persegi..."
}
```