[](https://2.bp.blogspot.com/-XnIU2Tsg3ak/VrcGX2um8bI/AAAAAAAADIA/LKuaf5WXPR8/s1600/KARAKTERISTIK%2BOOP.jpg)

Di dalam penggunaan konsep pemrograman yang berbasis objek atau yang disebut Object Oriented Pemrograman (OOP), haruslah memiliki karakteirstik. Adapun karakteristik tersebut adalah memiliki sifat turunan atau pewarisan (Inheritance), satu nama memiliki banayak sifat atau perilaku (Polymorphism), pembungkusan sifat dari objek yang berbeda (Encapsulation). Berikut akan dijelaskan karakteristik OOP tersebut:

Baca juga : [Pengertian dan Keuntungan Object Oriented Programming OOP C++](http://belajarcpplus.blogspot.com/2016/01/pengertian-dan-keuntungan-object-oriented-programming-oop-cplusplus.html)

**Karakterisitik Object Oriented Pemrograman (OOP)**

**Inheritance**

Inheritance memungkinkan programmer untuk "menurunkan" sebuah class menjadi class lain yang lebih spesifik dan memiliki fungsi yang lebih komplit. Inheritance merepresentasikan hubungan "seperti" atau "sejenis" (a kind of). Sebagai contoh, sebuah perahu motor adalah seperti perahu namun dengan kemampuan lebih, yakni memiliki motor. Contoh lain adalah kendaraan jenis mobil (sebagai super class) dan memiliki tipe sport (sebagai subclass), bila digabung menjadi mobil sport.

Ketika kita menurunkan sebuah class dari class yang lain, class yang baru akan mewarisi semua attribute dan method dari class yang sudah ada. Class yang sudah ada disebut dengan base class atau super class atau parent class dan class yang baru disebut dengan derived class atau subclass, atau child class. Dengan inheritance, kita dapat men daur ulang program kita atau bahkan men daur ulang program orang lain agar sesuai dengan kebutuhan kita.

Tanpa inheritance, kelas merupakan sebuah unit yang berdiri sendiri. Inheritance akan membentuk suatu konsep dimana jika konsep yang diatas berubah maka perubahan akan juga berlaku pada yang ada dibawahnya. Inherate sangat mirip dengan hubungan orang tua dengan anak. Manakala suatu kelas menerima warisan, semua anggota data dan fungsi juga akan menerima warisan, walalupun tidak semuanya akan dapat di akses oleh anggota fungsi dari kelas. Di dalam C++ penentuan akses pada inheritance ada tiga macam, yaitu :

1. Public

Penentuan akses berbasis public menyebabkan anggota dari public dari sebuah kelas utama akan menjadi anggota public kelas turunan dan menyebabkan juga anggota protect kelas utama menjadi anggota protect kelas turunan, namun untuk anggota kelas private tetap pada private kelas utama.

2. Private

Penentu akses berbasis private menyebabkan anggota dari anggota public dari kelas utama akan menjadi anggota protect kelas turunan, dan menyebabkan anggota dari kelas utama menjadi anggota protect kelas turunan, namun untuk anggota kelas private tetap pada private kelas utama.

3. Protected

Penentu akses berbasis protect menyebabkan anggota dari anggota protect dan public dari kelas utama akan menjadi anggota private dari kelas turunan. Anggota private dari kelas utama selalu menjadi anggota private kelas utama. Pada waktu mendeklarasikan suatu kelas, anda dapat menandai bahwa suatu kelas berasal dari mana, yaitu dengan tanda titik dua (:) setelah nama kelas, tipe asalnya bias berupa public atau yang lainnya dan dari kelas mana berasal.

Contoh Bentuk Umum Penulisan :

#include <iostream>

#include <conio>

class Values

{

   protected: // dapat di gunakan di class turunannya nanti

   int P, L;

   public:

   void Nilai(int a, int b)

   { P=a; L=b; }

};

// class ini turunan dari class values

class luasPersegiPanjang: public Values

{

   public:

   int luas( )

   { return (P\*L); }

};

// class ini turunan dari class values

class kelilingPersegiPanjang: public Values

{

   public:

   int keliling( )

   {

      return((P +L) \* 2);

   }

};

int main ( )

{

   luasPersegiPanjang a; //pembuatan objek luasPersegiPanjang

   //pembuatan objek kelilingPersegiPanjang

   kelilingPersegiPanjang b;

   a.Nilai(4,5); // input nilai ke dalam method Nilai

   b.Nilai(4,5);

   cout <<"Luas Persegi Panjang = "<<a.luas( )<<endl;

   cout <<"Keliling Persegi Panjang = "<<b.keliling( )<<endl;

   getch( );

}

**Polymorphism**

Polymorphisms adalah kemampuan 2 buah object yang berbeda untuk merespon pesan permintaan yang sama dalam suatu cara yang unik. Contoh, saya melatih lumba-lumba saya dengan perintah untuk meloncat dan juga saya melatih burung untuk merespon perintah saya untuk berkicau. Saya lakukan latihan untuk merespon kepada mereka dengan perintah lisan. Melalui polymorphism saya tahu bahwa anjing dan burung akan merespon dengan gonggongan atau kicauan.

Baca juga : [Pengertian dari Compiler (Kompilator) C++](http://belajarcpplus.blogspot.co.id/2016/02/pengertian-dari-compiler-kompilator-cplusplus.html)

Dengan cara dan kemampuan seperti ini, jenis akses tertentu dapat diterapakan dengan berbagai cara oleh objek-objek yang memiliki tipe yang berbeda atau memiliki banyak bentuk. Fungsi virtual merupakan dasar dari polymorphism yang berkerja hanya dengan pointer-pointer dan referensi dan hanya mendeklarasikan method sebagai virtual. Adapun aturan dari virtual function sebagai berikut:

* Virtual function harus anggota class.
* Anggota class bukan anggota yang bersifat statis.
* Anggota class dapat diakses dengan pointer objek.
* Virtual function tidak dapat memiliki virtual constructor, akan tetapi dapatberupa virtual destructor.

Contoh Penggunaan Polymorphism

#include <iostream.h>

#include <conio.h>

class HewanPeliharaan

{

   public:

   void lucu( )

   {

      cout<<" Lucunya hewan peliharaan"<<endl;

   }

   virtual void makan( ) // konsep polymorphisme

   {

      cout<<" Makan... dibutuhkan hewan peliharaan"<<endl;

   }

};

class Jinak : public HewanPeliharaan

{

   public:

   void lucu( )

   {

      cout<<" Lucu dan Jinak"<<endl;

   }

   virtual void makan( ) // konsep polymorphisme

   {

      cout<<" Diberi makan agar jinak"<<endl;

   }

};

class Kucing : public Jinak

{

   public:

   void lucu( )

   {

      cout<<" Lucunya kucing"<<endl;

   }

   virtual void makan( ) // konsep polymorphisme

   {

      cout<<" Makanan dibutuhkan Kucing"<<endl;

   }

};

void main( )

{

   //definisi Objek Jinak dan Kucing puma

   Jinak jinak;

   Kucing puma;

   //definisi pointer ke objek

   HewanPeliharaan \*hewan;

   hewan=&jinak;

   hewan->lucu( );

   hewan->makan( );

   cout<<" -----------------------------"<<endl;

   hewan = &puma;

   hewan->lucu( );

   hewan->makan( );

   getch( );

}

**Encapsulation**

Ciri penting lainnya dari OOP adalah encapsulation. Encapsulation adalah sebuah proses dimana tidak ada akses langsung ke data yang diberikan, bahkan hidden. Jika ingin mendapat data, maka harus berinteraksi dengan objek yang bertanggung jawab atas dara tersebut. Berikut cirri dari encapsulation:

1. Variabel dan method dalam suatu obyek dibungkus agar terlindungi

2. Untuk mengakses, variabel dan method yang sudah dibungkus tadi perlu interface

3. Setelah variabel dan method dibungkus, hak akses terhadapnya dapat ditentukan.

4. Konsep pembungkusan ini pada dasarnya merupakan perluasan dari tipe data Struktur

Baca juga : [Pengertian dan Fitur-fitur C++](http://belajarcpplus.blogspot.co.id/2016/01/pengertian-dan-pengenalan-fitur-fitur-c-plusplus.html)

Dua hal dalam enkapsulasi :

1. Information hiding

2. Menyediakan perantara (method) untuk mengakses data

Pada intinya, encapsulation adalah pemisahan antara bagian private dan public pada sebuah objek. Atau, dapat dipandang sebagai pemisahan antara interface (bagian private) dan implementation (bagian public). Objek-objek lain yang hendak berinteraksi dengan objek ini akan mengirimkan sebuah pesan (message) dan objek ini akan mengerjakan sesuatu dan mengirimkan pesan balik sebagai jawaban jika diperlukan.

Keuntungan utama dari encapsulation tentu saja adalah penyembunyian implementasi (implementation hiding). Dengan implementation hiding, kita dapat memperbaiki bagaimana objek kita bekerja tanpa harus khawatir bagaimana menginformasikan perubahan tersebut ke objek-objek yang lain. Selama kita tidak merubah interface dari objek kita, objek-objek yang lain akan tetap dapat menggunkan objek kita. Contoh penggunaan encapsulation

#include <conio.h>

#include <iostream.h>

class satu\_kandang

{

   int kandang;

   public:

   void atur(int nilai);

   int hasil(void);

};

void satu\_kandang :: atur(int nilai)

{

   kandang = nilai;

}

int satu\_kandang :: hasil(void)

{

   return kandang;

}

void main( )

{

   satu\_kandang ayam1, ayam2, ayam3;

   int bebek;

   ayam1.atur(5);

   ayam2.atur(7);

   ayam3.atur(2);

   bebek=20;

   cout<<" Nilai dari ayam1 adalah "<<ayam1.hasil( )<<endl;

   cout<<" Nilai dari ayam2 adalah "<<ayam2.hasil( )<<endl;

   cout<<" Nilai dari ayam3 adalah "<<ayam3.hasil( )<<endl;

   cout<<" Nilai dari bebek adalah "<<bebek<<endl;

   getch( );

}

Baca juga : [Pengertian dan Deklarasi Variabel C++](http://belajarcpplus.blogspot.co.id/2016/01/pengertian-dan-deklarasi-variabel-cplusplus.html)

Selain contoh diatas, penggunaan encapsulation dapat pula dengan membedakan fungsi methode. Ada deklarasi methode untuk input nilai dan ada pula yang di gunakan untuk output nilai. Contoh penulisan listingnya sebagai berikut :

Contoh

#include <iostream>

#include <conio>

class enkapsulasi

{

   public :

   char \*Nama;

   char \*NIM;

   int nilaiku;

   void setNilai(int nilai)

   { // methode yang digunakan untuk input

      nilaiku=nilai;

   }

   int getNilai( )

   { // methode yang digunakan untuk output

      return nilaiku;

   }

};

int main( )

{

   enkapsulasi data; // pembuatan objek class

//input datanya

   data.Nama = "Bima Chanindra";

   data.NIM = "12120001";

   data.setNilai(100);

//output datanya

   cout << "Nama = " << data.Nama << endl;

   cout << "NIM = " << data.NIM << endl;

   cout << "Nilai = " << data.getNilai( );

   getch( );

}

Baca juga : [Macam Macam dan Contoh Operator C++](http://belajarcpplus.blogspot.co.id/2016/01/macam-macam-dan-contoh-operator-cplusplus.html)

Cukup sekian dari belajarcpp.com semoga artikel ini bisa bermanfaat untuk kita semua. Mohon maaf atas kekuranganya dan Terima Kasih atas dukungan dan kunjungan anda ke BelajarCPP. Have a nice day.

Last update : 6 Februari 2016