PEMROGRAMAN BERORIENTASI OBJEK

KONSEP PBO

Telkom University

Budhi Irawan, S.Si, M.T

PENDAHULUAN

- Meskipun bahasa C adalah bahasa pemrograman yang ***berkekuatan tinggi (powerful)*** atau bisa digunakan untuk membuat program apa saja, tapi ***bahasa C tidak mendukung adanya pemrograman berorientasi objek***.
- Bahasa C masih merupakan ***prosedural,*** salah satu alasan ***Bjarne Stroustrup*** menciptakan bahasa C++ adalah untuk menambahkan ***kemampuan pemrograman berorientasi objek (PBO)*** kedalam bahasa C sehingga mula-mula dia menamai C++ sebagai bahasa **"C dengan Kelas"**

PEMROGRAMAN BERORIENTASI OBJEK

- Perlu diperhatikan bahwa ***PBO bukanlah sebuah bahasa pemrograman*** melainkan sebuah cara atau metodologi yang digunakan agar program yang kita buat menjadi lebih ***modular*** karena suatu permasalahan akan dikumpulkan dalam ***satu objek***.
- Dalam kode program yang dibuat, suatu ***objek*** akan diterjemahkan kedalam ***bentuk kelas***.

C++ DAN PBO

- ***Bahasa C++*** merupakan bahasa yang telah diakui keunggulannya, selain itu sudah sangat populer dan banyak digunakan diseluruh dunia.
- ***Bahasa C++*** merupakan bahasa C yang ditambahkan kemampuan ***PBO***, maka dari itu, C++ tentu menjadi bahasa yang istimewa.
- Ini berarti bahwa dengan menggunakan C++ maka dapat dikembangkan suatu program yang menggunakan fitur-fitur ***PBO***, tanpa harus meninggalkan kemampuan-kemampuan hebat yang dimiliki oleh bahasa C.

PERBEDAAN PEMROGRAMAN PROSEDURAL DENGAN PBO

- ***PBO*** merupakan bentuk penyederhanaan dari prosedural sehingga program akan lebih mudah dikembangkan.
- Dalam bahasa prosedural, untuk menyelesaikan salah satu permasalahan dalam program, kita harus *membuat banyak fungsi* yang tentunya akan memakan waktu dan konsentrasi.
- Dengan kata lain, dalam ***bahasa prosedural akan lebih difokuskan dengan "Bagaimana Cara Membuat?", bukan "Apa yang Akan Dibuat?"***

PERBEDAAN PEMROGRAMAN PROSEDURAL DENGAN PBO

- Berdasar kepada pengalaman, satu buah program / aplikasi sistem yang kompleks ditulis menggunakan bahasa C rata-rata berisi 25.000 s/d 100.000 baris kode.
- Hal ini tentu akan menyebabkan program tersebut menjadi sangat rumit dan susah untuk dipahami alurnya.
- Karena alasan inilah bahasa C++ diciptakan dengan tujuan dapat ***menyederhanakan*** program tersebut dengan cara ***memecahnya*** kedalam ***sub-sub program*** yang dinamakan dengan ***"kelas"***.

PERBEDAAN PEMROGRAMAN PROSEDURAL DENGAN PBO

- ***"Kelas"*** itu sendiri kemudian dapat digunakan kembali dalam pembuatan program lain tanpa harus melakukan ***pengkodean ulang***.
- Melalui cara ini tentu kita dapat lebih berkonsentrasi ke arah pembentukan program sehingga ***waktu yang dibutuhkan juga akan lebih cepat***.

PERBEDAAN PEMROGRAMAN PROSEDURAL DENGAN PBO

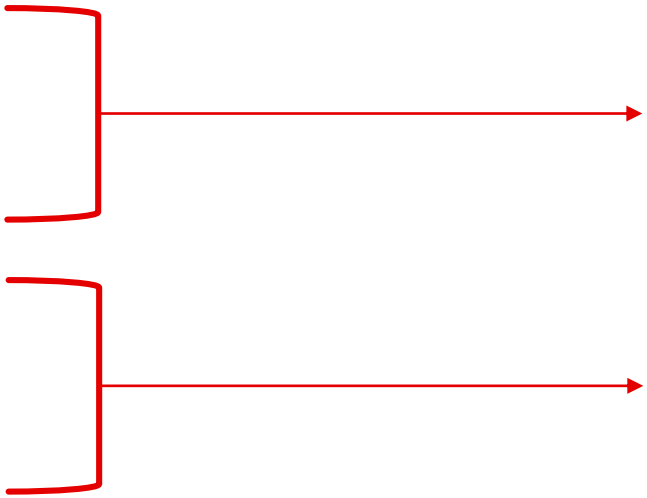
- Meskipun demikian, tingkat kesulitan yang biasanya akan muncul dalam bekerja dengan ***PBO*** adalah pada ***saat perancangan program***.
- Walaupun dari segi konsep PBO merupakan materi yang mudah dipahami, tapi pada kenyataannya ***PBO akan susah diimplementasi kan***.
- Bagi para pemula, pada umumnya mereka akan kesulitan dalam menerjemahkan suatu permasalahan kedalam ***bentuk "kelas"***.

PERBEDAAN PEMROGRAMAN PROSEDURAL DENGAN PBO

PEMROGRAMAN PROSEDURAL

```
data1
data2
…
fungsi1()
fungsi2()
…
dataA
dataB
…
fungsiA()
fungsiB()
…
main() (fungsiutama)
```

PBO




```
main() (fungsiutama)
```

KARAKTERISTIK PBO

- Semua bahasa pemrograman yang mendukung ***PBO*** harus mengimplementasikan konsep ***"abstraksi"***, ***"pembungkusan"***, ***"pewarisan sifat"***, dan ***"polimorfisme"***.
- Konsep-konsep tersebut merupakan ciri atau karakteristik utama dari bahasa yang mendukung ***PBO***

I. ABSTRAKSI (ABSTRACTION)

- **"Abstraksi"** adalah proses ***"pengabstrakan"*** atau ***"penyembunyian"*** detail program yang sangat rumit sehingga kita tidak perlu untuk mempermasalahkan pembuatannya.
- Kita hanya perlu ***"objek"*** tersebut dapat kita gunakan sesuai fungsinya, sebagai contoh lihatlah objek disekitar kita misalnya **mobil**, pembuat mobil tidak perlu mendefinisikan cara *pembuatan* maupun *komponen-komponen* yang diperlukan untuk merakit satu unit mobil.

1. ABSTRAKSI (ABSTRACTION)

- Disini berarti bahwa pembuat mobil telah ***"mengabstraksikan"*** proses dan data yang terdapat didalamnya.
- Yang penting bagi kita (sebagai pengendara) adalah mobil tersebut dapat kita gunakan sebagaimana mestinya.
- Hal ini tidak berbeda dengan abstraksi suatu fungsi yang terdapat pada sebuah kelas didalam program.

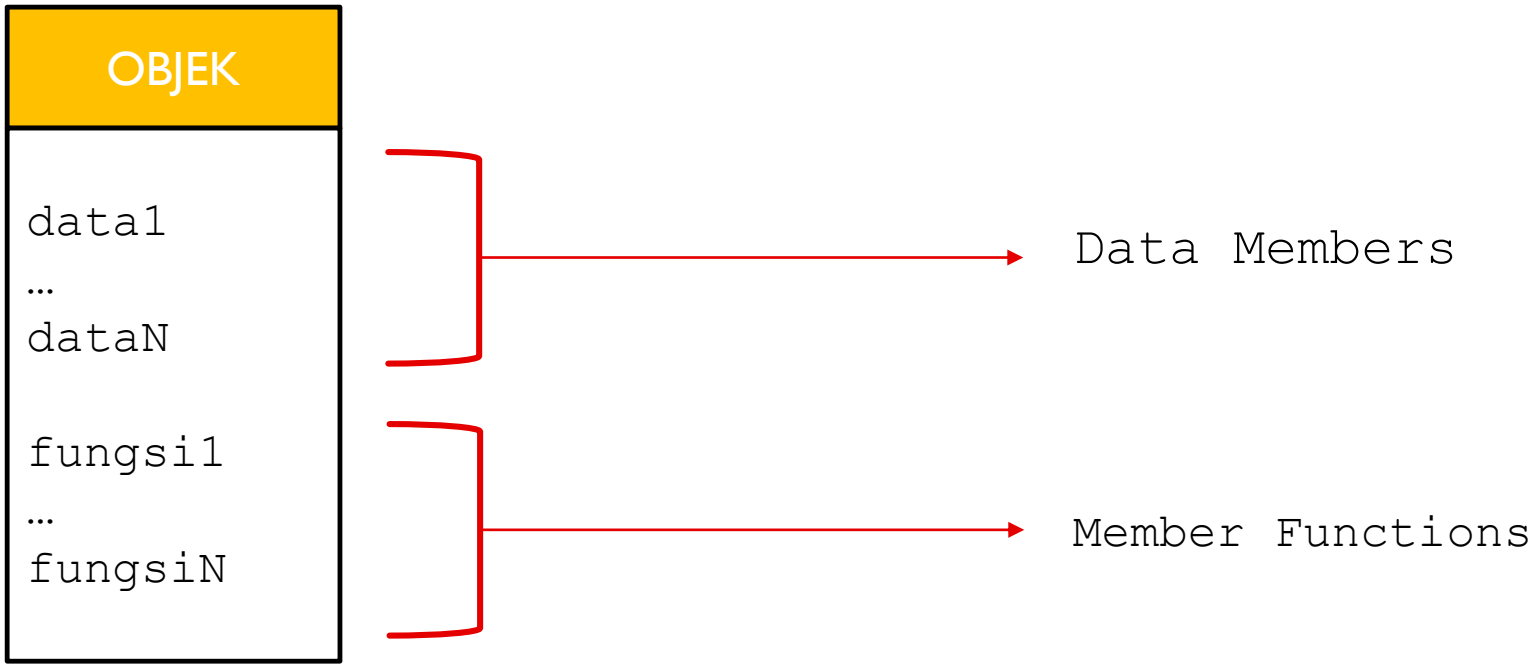
2. PEMBUNGKUSAN (ENCAPSULATION)

- ***"Pembungkusan"*** yang dimaksud disini adalah sebuah proses ***"penggabungan"*** antara ***data*** dan ***fungsi-fungsi*** yang berkaitan kedalam sebuah ***"objek"***.
- Dengan demikian kita dapat membuat program yang terintegrasi tanpa harus mendeklarasikan ***variabel-variabel*** yang bersifat ***eksternal***.

2. PEMBUNGKUSAN (ENCAPSULATION)

- Dalam C++, data (bisa berupa variabel atau konstanta) dalam sebuah ***"kelas"*** disebut dengan ***"data members"***, sedangkan ***fungsi-fungsi***nya dikenal dengan istilah ***"member function"***.
- Dalam bahasa pemrograman lain yang mendukung **PBO**, ada yang menyebut ***data*** *dalam sebuah* ***kelas*** dengan istilah ***"field"*** maupun ***"properti"*** dan Fungsi-fungsi yang terdapat didalam ***"kelas"*** disebut sebagai ***"method"***.

2. PEMBUNGKUSAN (ENCAPSULATION)



3. PEWARISAN (INHERITANCE)

- Dalam ***PBO***, kita dapat menciptakan ***kelas baru*** yang ***"diturunkan"*** dari kelas lain yang sudah didefinisikan sebelumnya.
- ***Kelas baru*** ini sering disebut dengan ***"kelas turunan (derived class)"*** sedangkan ***objek induknya*** disebut dengan "***kelas dasar (base class)***".
- Dalam bahasa ***java, kelas dasar*** disebut "***superclass***" dan ***kelas turunan*** disebut "***subclass***", dalam Delphi, ***kelas dasar*** disebut ***"ancestor"*** dan ***kelas turunannya*** disebut "***descendant***".

3. PEWARISAN (INHERITANCE)

- ***Sifat yang terkandung*** pada ***objek turunannya*** adalah ***sifat hasil pewarisan dari sifat-sifat yang terdapat pada objek induk***.
- Maka dari itu, proses seperti ini sering dikenal dengan istilah **"*pewarisan (inheritance)*".**
- Dengan fitur seperti ini, kita dapat membuat ***objek baru*** yang memiliki ***kemampuan lebih*** dibanding objek ***induknya***, yaitu dengan cara menambahkan ***sifat-sifat baru*** kedalam **"*objek*"** tersebut.

KELAS

- ***Kelas*** merupakan ***"cetak biru (blue print)"*** atau "***prototipe/kerangka"*** yang mendefinisikan ***variabel-variabel (data) dan fungsi-fungsi (perilaku) umum*** dari sebuah ***objek*** tertentu.
- Sebagai contoh kita ambil ***objek mobil*** yang mana mobil memiliki data seperti ***warna, tahun, merk, tipe, nomor polisi*** dan sebagainya.
- Selain data ciri-ciri pisik tersebut, mobil juga memiliki perilaku-perilaku spesifik yang dapat membedakan antara mobil yang satu dengan yang lainnya seperti : ***sistem pengereman, perubahan roda gigi (persneling)*** dll

KELAS

- Sekarang kita ambil contoh ***"objek"*** lain, misalnya manusia yang memiliki data atau ciri-ciri isik seperti : ***nama, tinggi dan berat badan, bentuk sidik jari, model rambut*** dll.
- Selain itu manusia memiliki prilaku-prilaku seperti ***cara berjalan, cara berbicara***, dll
- Dalam program, objek-objek seperti ini dapat didefinisikan sebagai sebuah ***"kelas"***.

KELAS

- Dalam dunia pemrograman sebenarnya ***kelas*** tidak jauh berbeda dengan ***tipe data sederhana*** seperti yang telah kita bahas pada materi sebelumnya.
- Perbedaanya, tipe data sederhana digunakan untuk *mendeklarasikan* ***variabel "normal"*** *sedangkan "****kelas****" digunakan untuk mendeklarasikan sebuah* ***variabel yang berupa objek***.

KELAS

- Penting untuk diperhatikan bahwa ***"kelas"*** masih bersifat ***"abstrak"***, yang mana pada saat kelas baru dibuat maka telah didefinisikan ***tipe data baru***.
- Sekali didefinisikan maka ***tipe data baru*** ini dapat digunakan untuk membuat suatu ***"objek"*** dari tipe tersebut.
- Dengan kata lain ***"kelas"*** adalah ***pola (template)*** untuk ***pembuatan "objek", dan "objek" adalah wujud nyata (instance) dari sebuah kelas***.

KELAS

- Sebagai contoh manusia adalah ***"kelas"***; sedangkan ***"objek"*** atau ***"wujud nyata"*** dari ***"kelas manusia"*** adalah iwan, tono, soni dll. Contoh :

```
// mendefinisikan kelas dengan nama manusia
class Manusia{
    // data dan fungsi
    // …
};

// Mendeklarasikan objek (instance)bertipe manusia
manusia iwan, tono, soni;
```

MEMBUAT KELAS

- Dalam bahasa C++, ***"kelas"*** dibuat dengan menggunakan keyword **`class`**. Adapun bentuk umum penulisannya adalah sbb :

```
class nama_kelas{
    access_specifier1:
        data_members;
        member_functions;
        …
    access_specifier2:
        data_members;
        member_functions;
        …
    …
};
```

MEMBUAT KELAS

- Untuk mendefinisikan atau membuat implementasi fungsi-fungsi yang terdapat dalam sebuah ***"kelas"***, kita menggunakan **operator ::** Berikut ini adalah bentuk umum penulisannya :

```
tipe_data nama_kelas::nama_fungsi(daftar_parameter){
    statemen_yang_akan_dilakukan;
    …
}
```

MEMBUAT KELAS

- Selanjutnya, untuk mengakses ***data*** atau ***fungsi*** yang terdapat didalam **"*kelas*"** tersebut, kita menggunakan tanda ***titik***. Berikut ini bentuk umum dari proses pengaksesan data atau fungsi dari sebuah kelas :

```
nama_instance.data
```

atau

```
nama_instance.nama_fungsi(daftar_parameter)
```

MEMBUAT KELAS

```
#include <iostream>
using namespace std;
class bangun{
    private:
    float x,y;
    public:
    float luas(){return x*y;};
    void beri_Nilai(float panjang, float lebar);
};
//scope operator (::)
void bangun::beri_Nilai(float panjang, float lebar){
   x = panjang;
   y = lebar;
}
int main(){
   bangun a,b;
   a.beri_Nilai(4,3);
   b.beri_Nilai(7,5);
   cout<<"Luas persegi panjang a adalah = "<<a.luas()<<endl;
   cout<<"Luas persegi panjang b adalah = "<<b.luas()<<endl;
   return 0;
}
```

CONSTRUCTOR DAN DESTRUCTOR

- ***Constructor*** adalah sebuah ***fungsi*** yang otomatis akan dipanggil setiap kali melakukan ***"instansiasi"*** nilai dari data-data yang terdapat didalam ***"kelas"*** bersangkutan.
- Sama halnya seperti ***fungsi biasa***, pada ***contructor*** juga dapat ditambahkan ***parameter*** ataupun dilakukan ***overload***.
- Namun perlu diperhatikan bahwa nama dari ***fungsi constructor*** harus sama dengan ***nama kelasnya*** dan ***tidak memiliki tipe kembalian*** (tidak juga void)

CONSTRUCTOR

```
#include <iostream>
using namespace std;
class bangun{
     private:
     float x,y;
     public:
     float luas(){return x*y;};
     bangun(float panjang, float lebar);
};
//konstruktor
     bangun::bangun(float panjang, float lebar){
     x = panjang;
     y = lebar;
}
int main(){
     bangun a(4,7);
     bangun b(5,6);
     cout<<"Luas persegi panjang adalah = "<<a.luas()<<endl;
     cout<<"Luas persegi panjang adalah = "<<b.luas()<<endl;
return 0;
}
```

CONSTRUCTOR

- Pada program di atas, diimplementasikan **fungsi *beri_Nilai*** dengan menggunakan ***konstruktor berparameter*** yaitu **`panjang`** dan **`lebar`** yang nantinya akan diumpan ke **variabel `x`** dan **`y`**.
- Sekarang bisa digunakan ***konstruktor*** ini dengan memanggil ***objek*** dari ***kelas bangun*** disertai parameter terkait dengan ***konstruktor*** yang ada.

DESTRUCTOR

- ***Destrucor*** adalah fungsi yang merupakan kebalikan dari ***constructor***, yaitu berguna untuk ***menghancurkan*** atau ***membuang*** sebuah ***objek (instance)*** dari ***memori***.
- ***Fungsi*** ini juga akan dipanggil secara otomatis ketika program telah selesai dijalankan.
- Nama dari ***fungsi destructor*** adalah sama seperti nama ***"kelas"*** maupun ***nama constructor***, tetapi didepannya ditambahkan tanda ***tilde ('~').***
- ***Destructor*** juga ***tidak memiliki tipe kembalian*** (tidak juga void).

DESTRUCTOR

- Karena tidak adanya deklarasi pada saat menghapus objek, maka destruktor tidak memiliki parameter.
- Berikut contoh sintaks sinkat penggunaan destruktor guna me-*reset* x dan y setelah objek berakhir masa kerjanya.

```
bangun::~bangun(){
x = 0;
y = 0;
}
```

- ketika suatu objek telah melaksanakan tugasnya, maka secara otomatis destruktor dipanggil.

DESTRUCTOR

```
#include <iostream>
using namespace std;
class bangun{
      private:
      float x,y;
      public:
      float luas(){return x*y;};
      bangun(float panjang, float lebar);
      ~bangun();
};
//konstruktor
bangun::bangun(float panjang, float lebar){
      x = panjang;
      y = lebar;
}
//destruktor
bangun::~bangun(){
      cout<<"Reset"<<endl;
      x=0;
      y=0;
}
int main(){
      bangun a(4,7);
      bangun b(5,6);
      cout<<"Luas persegi panjang adalah = "<<a.luas()<<endl;
      cout<<"Luas persegi panjang adalah = "<<b.luas()<<endl;
      return 0;
}
```

TINGKAT AKSES

- Dalam C++, penentu hak akses disebut dengan ***acces specifier***, artinya akan terdapat pembatasan akses terhadap ***data*** maupun ***fungsi*** tersebut sehingga tidak semua ***"kelas"*** maupun lingkungan program dapat bebas mengaksesnya.
- Dalam C++, terdapat ***3 buah tingkat akses*** yang dapat digunakan untuk menentukan bagaimana ***data*** maupun ***fungsi*** didalam suatu kelas dapat diakses dari lingkungan luar yaitu ***"public", "private"*** dan ***"protected.***

I. PRIVATE

- ***Tingkat akses*** ini berguna untuk memberikan hak akses data hanya kepada ***"kelas"*** yang bersangkutan saja, artinya ***kelas-kelas turunan*** ataupun lingkungan luar didalam program ***tidak diizinkan*** untuk mengakses data tersebut.
- Dalam C++, untuk menentukan data tersebut bersifat **"*private*"**, maka kita harus menggunakan kata kunci **"*private*"**,.
- Secara default, jika tidak menuliskan tingkat akses dalam pendeklarasian data (maupun fungsi) dalam sebuah **"*kelas*"**, maka data atau fungsi tersebut akan dianggap sebagai data **"*private*"**.

I. PRIVATE

```
#include <iostream>

using namespace std;

class CONTOH {
  int X;      // X bersifat private
public:
  void SetX(int XX) {
    X = XX;
  }
  void ShowX() {
    cout<<"Nilai X: "<<X<<endl;
  }
};

// Fungsi utama
int main() {

  CONTOH O;
  O.SetX(100);
  O.ShowX();

  return 0;
}
```

I. PRIVATE

- Seperti dilihat diatas bahwa X bersifat ***private*** dan dia hanya dapat diakses oleh fungsi-fungsi yang terdapat didalam kelas CONTOH.
- Dengan kata lain, lingkungan luar maupun kelas-kelas lain *tidak diizinkan untuk mengakses X*

2. PUBLIC

- Tingkat akses ini berguna *untuk memberikan hak akses secara umum atau publik kepada* ***kelas-kelas turunannya*** *maupun terhadap* ***lingkungan luar*** *didalam program.*
- Bagian ini biasanya berisi ***fungsi-fungsi*** yang akan dijadikan sebagai ***penghubung (interface)*** dari bagian ***"private"*** suatu kelas dengan ***lingkungan luar***.
- Untuk menjadikan data maupun ***fungsi*** dengan sifat ***"publik",*** kita perlu menggunakan kata kunci "***public***".

2. PUBLIC

```
#include <iostream>

using namespace std;

class CONTOH {
  int X;
public:
  void SetX(int XX) {
    X = XX;
  }
  int KuadratX() {
    return X * X;
  }
  // ...
};

// Fungsi utama
int main() {

  CONTOH O;
  int hasil;

  // Melakukan pemanggilan thdp fungsi-fungsi yg terdapat di dlm kelas CONTOH
  O.SetX(10);
  hasil = O.KuadratX();
  cout<<"Hasil: "<<hasil;

  return 0;
}
```

2. PUBLIC

- Seperti yang kita lihat bahwa kita dapat memanipulasi nilai X (yang bersifat ***private***) dari lingkungan luar melalui ***fungsi-fungsi*** penghubung yang ditempatkan pada bagian ***public*** (yaitu fungsi SetX() dan KuadratX())

3. PROTECTED

- Tingkat akses ini *digunakan untuk memberikan hak akses terhadap data (maupun fungsi) dalam suatu* ***kelas*** *sehingga data tersebut dapat diakses* ***oleh kelas turunannya.***
- Namun, lingkungan luar didalam program masih tetap tidak diberi hak untuk mengaksesnya.
- Untuk menyatakan data maupun fungsi dengan sifat seperti ini maka perlu menggunakan ***keyword protected.***

3. PROTECTED

```
##include <iostream>

using namespace std;

// Membuat kelas DASAR yang didalamnya terdapat data bersifat protected

class DASAR {
protected:
  int X;
public:
  // Inisialisasi nilai X dengan nilai 10
  DASAR() { X = 10; }
  //...
};

// Membuat kelas TURUNAN yang merupakan turunan dari kelas DASAR
class TURUNAN: public DASAR {
  int Y, hasil;
public:
  void SetY(int YY) {
    Y = YY;
  }
  void KaliXY() {
    // Menggunakan nilai X dari kelas DASAR
    hasil = X * Y;
  }
  int GetHasil() {
    return hasil;
  }
};

// Fungsi utama
int main() {

  DASAR A;
  TURUNAN B;

  B.SetY(5);
  B.KaliXY();
  cout<<"\nHasil X kali Y: "<<B.GetHasil();

  return 0;
}
```