FENOMENA ALAM DI DALAM DIMENSI PEMBUATAN PROGRAM DENGAN MENGGUNAKAN *OBJECT ORIENTED PROGRAMMING*

Ditdit N. Utama
Dosen FASILKOM - UIEU
ditdit.nugraha@indonusa.ac.id

Abstrak

Empat era pengembangan program sudah di lewati di lingkungan aktivitas para praktisi Teknologi Informasi. Walau pun dua era terakhir masih mencari jati dirinya, artinya hanya beberapa perusahaan pengembang Sistem saja yang mau mengembangkan sistem informasinya dengan menggunakan *Object Oriented Programming* atau *Component Oriented Programming*. Dimensi, cara pandang dan perubahan budaya pengembangan sistem informasi menjadi alasan utama mengapa para pengembang masih merasa resist untuk menggunakan metode atau konsep yang mengusung kemudahan ini. Disini akan dibahas mengenai fenomena alam di dalam dimensi dan cara pandang pengembangan program dan sistem informasi dengan menggunakan *Object Oriented Programming Concept*.

Kata Kunci: Era Pengembangan P*rogram, Object Oriented Programming, Component Oriented Programming,* Fenomena Alam

Pendahuluan

Sudah empat era yang terlewati dalam pengembangan sistem informasi (*program*) di dunia Teknologi Informasi ini. Keempat era ini adalah:

1. Konsep Tradisional
   Dalam metode ini, pengembangan program hanya diorientasi pada penyimpanan file secara fisik. Program yang dihasilkan dari pengembangan program dengan menggunakan konsep tradisional ini tidak beranjak pada kebutuhan akan informasi.
2. Konsep Terstruktur
   Konsep ini lebih baik dari konsep sebelumnya. Konsep terstruktur ini lahir di awal tahun 1977-an. Konsep ini sudah memperhatikan kebutuhan akan informasi dan hubungan antar informasi yang ada dalam pengembangan programnya. *Data Flow Diagram* (DFD) dan *Entity Relationship Diagram* (ERD) adalah diagram-diagram andalan dalam konsep pengembangan sistem ini.
3. Konsep Orientasi Objek
   Konsep ini beranjak dari pemahaman bahwa informasi ada pada satu kesatuan entitas yang disebut dengan objek. Pengembangan program dengan menggunakan konsep ini lebih memperhatikan aspek-aspek fenomena alam dari pada kebutuhan

informasi dan atau *information flow*.

4. Konsep Orientasi Komponen
   Era keempat ini membuat para praktisi IT lebih dininabobokan dalam pengembangan program. Program sudah di-cluster menjadi komponen-komponen yang beraneka ragam, tinggal dikolaborasikan saja antar komponen tersebut.

Ada beberapa dimensi yang membedakan untuk masing-masing era tersebut. Konsep yang menjadi andalan di masing-masing eranya. Seperti yang terjadi dalam konsep Tradisional yang beranjak dari konsep pengembangan tradisional ini mengunakan kaidah-kaidah kontrol alur berupa; urutan, keputusan dan *loop*/pengulangan. Masih di dalam konsep tradisional juga, bahwa konsep ini pengembangan program hampir selalu dimulai dengan pemikiran file secara fisik, yang sama sekali tidak berorientasi pada kebutuhan informasi.

Pemahaman akan konsep pertama ini mengalami gradasi yang cukup besar. Karena jelas pemahaman konsep file, data dan informasi secara fisik, jauh dan lebih sulit dengan pemahaman konsep *file*, data dan informasi secara logik. Permasalahan ini bisa diatasi dengan menggunakan konsep era kedua (konsep terstruktur) dalam pengembangan programnya.

Dalam konsep tersetruktur ini, pengembangan program sudah dimulai dari pemahaman *file*, data dan informasi secara logik. Dan orientasi kebutuhan informasi pun sudah menjadi basis pemahaman dalam pengembangan program dengan menggunakan konsep terstruktur ini, walaupun stuktur kontrol alurnya masih sama dengan konsep tradisional sebelumnya.

Tinjauan Teori

Konsep yang lahir di tahun 1977-an ini menggunakan beberapa diagram sebagai metode penggambaran sistem/program yang dikembangkan oleh para pengembangnya. Sebut saja *Data Flow Diagram* (DFD); yaitu sebuah diagram yang menggambarkan alur dan arus data, informasi dan proses yang terjadi pada sistem informasi/program yang dikembangkan ini. Selain itu ada juga yang disebut dengan *Entity Relationship Diagram* (ERD); yaitu sebuah diagram yang mampu menggambarkan hubungan satu entitas dengan entitas lain dalam sebuah sistem informasi/*program* secara fisik.

Perkembangan *tools* pengembang sistem/program serta generator semakin hari semakin membuat praktisi pengembang sistem informasi/program menjadi lebih mudah. Dua era pengembangan sistem informasi/program yang terakhir ini adalah cerminan dari perkembangan tools yang ada. Tetapi kemudahan pengembangan program dengan menggunakan dua konsep terakhir ini belum banyak digunakan di kalangan praktisi IT. Karena, membuminya konsep terstruktur menjadi alasan utama.

Object Oriented Programming

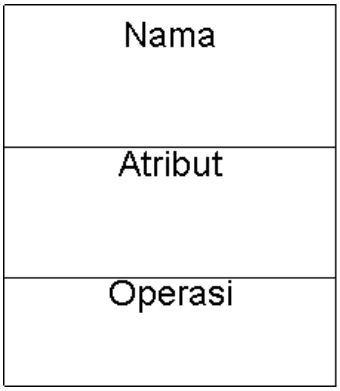
Object Oriented Programming merupakan sebuah konsep pemrograman yang diawali dengan memahami segala macam bagian program dengan konsep objek, dimana konsep objek ini merupakan

konsep alamiah yang diambil dari fenomena alam yang ada di dalam kehidupan sehari-hari manusia.

Objek dapat didefiniskan lepas, objek adalah sesuatu yang dapat dilihat atau diraba atau dirasa. Contoh dari objek secara cepat dapat disebuatkan, missal: udara, angina, planet, bintang, bangunan, organisasi, perusahaan, mahasiswa, dosen mata kuliah dan lain sebagainya. Yang jelas semua fenomena alam dapat digambarkan dengan menggunakan objek.

Object VS *Class*

Dalam perjalanannya, fenomena objek tidaklah hanya memiliki jumlah yang tunggal (kebanyakan). Misalnya planet. Planet adalah objek, tetapi planet tidak hanya berjumlah satu (yang pasti lebih dari satu), sehingga planet dalam konsep ini dapat dikatakan sebagai *Class*. Jadi sebenarnya *Class* adalah gabungan dari beberapa objek yang memiliki ciri dan karakteristik yang sama. Contoh lain Mahasiswa. Mahasiwa adalah objek, tetapi setiap mahasiswa pasti sama dalam hal memiliki NIM, Nama, Alamat dan lain sebagainya. Bahkan tingkah laku mahasiswa pun sama seperti kuliah, daftar ulang, bayaran dan lain sebagainya. Jadi di dalam konsep ini mahasiwa dapat juga disebut sebagai *Class*. *Class* dapat digambarkan secara logik dan memliliki bagian-bagian seperti berikut (gambar 1).



Sumber: Hasil pengolahan data

Gambar 1. Gambaran Logik *Class*

Misalkan contoh sebuah *Class* :

Nama :	*Class* Mahasiswa
Atribut :	Nim
	Nama
	Alamat
	Jenis_Kelamin
Operasi :	Kuliah
	Membayar
	Ujian
	Daftar_Ulang

Pembahasan

Konsep Objek dan *Class*

Dimensi selanjutnya mengenai objek dan *class* adalah konsepnya itu sendiri. Bahwa setiap objek/*class* pasti memiliki karakteristik seperti berikut :

1. *Encapsulation*

   Bahwa setiap objek/*class* bisa menutupi informasi atau data yang dimilikinya sendiri dari objek/*class* yang lain. Atribut dari sebuah *class* dapat berbentuk *public*, *private* dan *protected*. Jika berbentuk *public*, maka atribut atau data tersebut dapat diakses oleh *class* itu sendiri atau *class* di luarnya. Jika berbentuk *private*; maka data atau atribut itu hanya bisa diakses oleh *class* itu sendiri. Sedangkan jika data/ atribut sebuah *class* bertipe

protected, berarti data/atribut *class* tersebut hanya dapat diakses oleh *class* itu sendiri dan turunannya.

2. *Polymorphism*
   Sebuah objek atau *class* dapat berubah bentuk karena keadaan tertentu. Misal *Class* Mahasiswa, akan dapat berubah bentuk menjadi sebuah class baru, misalnya *Class* Mahasiswa Cuti, hanya karena atribut status kuliahnya berubah nilai.
3. *Inheritance*
   Bahwa objek atau *class* dapat memiliki sifat-sifat yang dimiliki objek atau class di atasnya. Atau objek atau *class* tersebut pun dapat menurunkan sifat-sifat pada objek/*class* yang ada di bawahnya. Atau dalam istilah *programming*-nya disebut sebagai perluasan *class* (*class extending*)

Kelebihan OOP

Ada beberapa kelebihan dan keunggulan dari konsep *Object oriented Programming* ini, diantaranya adalah :

1. Merupakan konsep yang umum yang dapat digunakan untuk memodel hampir semua *phenomena* dan dapat dinyatakan dalam bahasa umum (*natural language*)
2. Memberikan informasi yang jelas tentang *context* dari *system*
3. Mengurangi biaya *maintenance*
   a. Memudahkan untuk mencari hal yang akan diubah
   b. Membuat perubahan menjadi *local*, tidak bepengaruh pada modul yang lainnya
4. Kode yang telah dibuat dapat digunakan kembali (*reusability*)

Fenomena Alam Pada OOP

Perhatikan contoh penggalan program sederhana di bawah ini :

```
int main(){
        Buah b;

        b.setNama("Jeruk");

        std::cout<< b.getNama();

        return 0;
}
```

Kalau mengacu pada gambar *class* secara logik, penggalan program tersebut di atas sudah mencerminkan konsep *class* secara benar. Dimana *class* tersebut memiliki nama: buah; memiliki atribut : nama; serta memiliki operasi *setNama*() dan *getNama*(). Atribut *class* Buah di atas bertipe *private*, artinya atribut yang ada pada *class* Buah hanya dapat diakses oleh dirinya sendiri. Sedangkan operasi pada *class* Buah bertipe *public*, ini dapat diartikan bahwa fungsi-fungsi atau operasi yang ada pada *class* ini dapat diakses oleh dirinya sendiri dan *class* di luar dirinya.

Perhatikan fenomena alam lain pada contoh program yang lain:

```
#include <iostream>

class Hewan
{
private:
    char nama[30];
public:
    void setNama(char* nm)
    {
        strcpy(nama,nm);
    }
        char* getNama()
    {
        return nama;
```

```
    }
};

void main()
{
    Hewan h1, h2;
    h1.setNama("ayam");
    h2.setNama("lumba-lumba");
    cout << h1.getNama()<<endl
    Cout << h2.getNama()<<endl;
}
```

Fenomena alam dapat digambarkan di dalam penggalan program di atas. Pada program di atas, h1 dan h2 adalah contoh dari objek pada *class* Hewan.

Perhatikan pula penggalan program di bawah ini, bahwa konsep *polymorphism* dan *inheritance* dapat menggambarkan fenomena alam yang lain :

```
#include <iostream>
class Hewan
{
    public:
        virtual void sound()
        {
        }
};

class Anjing:public Hewan
{
    public:
        void sound()
        {
            cout << "guuk";
        }
};

class Kucing:public Hewan
{
    public:
        void sound()
        {
            cout << "meow";
        }
};

void main()
{
    Hewan* ph;
    ph = &Anjing();
    ph-> sound();
    ph = & Kucing();
    ph-> sound();
}
```

Dari program di atas, dapat dilihat bahwa fenomena alam (anjing dan kucing) sebagai hewan tercermin di contoh penggalan program di atas. Selain pendefinisian *class*, jelas juga bahwa *object oriented programming* dapat menggambarkan fenomena lain (operasinya) yang terdapat pada masing-masing *class* yang ada, dalam hal ini *class* kucing dan *class* anjing. *Fungsi sound ()* dapat berlaku pada *Class* Anjing atau Kucing, artinya konsep *polymorphism* berlaku dalam menggambarkan fenomena alam ini.

Polymorphism berarti "banyak bentuk", yang di dalam PBO berarti satu *member* fungsi, misal *sound*(), memiliki banyak bentuk prilaku. Misal, *sound*() untuk seekor anjing akan berbeda dengan seekor kucing.

Untuk tercapainya *polymorphism* ini harus dipenuhi beberapa syarat: (1) *derived class – derived class* yang dibentuk harus berasal dari satu *base class* yang sama, (2) member fungsi dideklarasikan *virtual*, (3) tipe *base class* harus berupa *pointer* atau *reference*.

Sedangkan bahwa anjing dan kucing adalah hewan (anjing dan kucing adalah turunan hewan),

adalah hasil dari konsep *inheritance* dalam menggambarkan fenomena alam ini.

Dalam konsep *inheritance*, Suatu *class* dapat dibuat dengan melakukan perluasan terhadap *class* yang lain. *class* yang diperluas sering disebut *base class*, sedangkan klas yang memperluas disebut *derived class*. *Derived class* melakukan spesialisasi terhadap *base class*.

Sedangkan konsep lain (*encapsulation*), dapat dijelaskan pada program di atas, bahwa hanya member data dan member fungsi dengan penspesifikasi akses ***public*** dan ***protected*** yang "diwariskan" dari *base class* ke *derived class*; dalam pengertian member tersebut dapat diakses secara langsung dari *derived class*.

OOP pun dapat memodelkan fenomena *friend* pada kehidupan dunia nyata. Perhatikan contoh penggalan program di bawah ini :

```
Class Mahasiswa
{   private :
    long Nomor_Induk;
    char Nama [50];
    char Jurusan [20];
public :
    Mahasiswa ()
    { Nomor_Induk = 0;
      strcpy (Nama, "");
      Strcpy(Jurusan, "");};

    void Inisialisasi (long No_Induk, char *Nama, char *Jurusan)
    {Nomor_Induk = No_Induk;
     strcpy (Nama, Nama);
     strcpy (Jurusan, Jurusan);};

    friend void Tampilkan_Data (Mahasiswa Mhs)
    { cout << "Nomor : " << Mhs.Nomor_Induk << endl;
    cout << "Nama: " << Mhs.Nama << endl;
    cout << "Jurusan : " << Mhs.Jurusan << endl;

void main ()
{   clrscr();
    Mahasiswa Mhs;
    Mhs.Inisialisasi (78939, "Budi", "Sistem Informasi");
    Tampilkan_Data (Mhs);}

class HasilUjian
    { private :
        long Nomor_Induk;
        float Nilai;
      public :
        HasilUjian (long Nomor, float Hasil) : Nomor_Induk (Nomor), Nilai (Hasil)
      {     };
friend float Nilai_Terbesar (HasilUjian a, HasilUjian b, HasilUjian c)
{   float Maks = a.Nilai;
Maks = (b.Nilai > Maks) ? b.Nilai : Maks;
Maks = (c.Nilai > Maks) ? c.Nilai : Maks;};}

void main ()
{   clrscr();
    HasilUjian Amir (8374, 78);
    HasilUjian Endah (7843, 90);
    HasilUjian Siti (7435, 98);
    cout << "Nilai Ujian Terbesar = " << Nilai_Terbesar (Amir, Endah, Siti);
}
```

Pengkapsulan memberikan keuntungan terutama memudahkan dalam perawatan program dan juga dalam

hal pencarian kesalahan. Namun, pengkapsulan untuk objek yang kompleks terkadang sulit dilakukan, sehingga seperti pada dunia nyata, adakalanya aturan ada yang dilanggar.

Fungsi *Friend* adalah fungsi yang bukan anggota *class* yang dapat mengakses anggota *class*. Fungsi seperti ini dapat mengakses anggota *class* baik yang bersifat *Private* maupun *Protected*.

Kesimpulan

Sebuah kemudahan yang ditawarkan oleh konsep *Object Oriented Programming* adalah bahwa konsep ini menawarkan kemudahan dalam memformulasikan/ mengkodekan hampir semua atau seluruh fenomena alam yang ada di sekitar manusia.

Semua konsep *object* pada *object oriented programming* (*polymorphism*, *inheritance* dan *polymorphism*) adalah terilhami dari fenomena alam yang ada di dunia nyata.

Kemudahan lain adalah bahwa perkembangan *tools* yang menunjang dalam pembuatan *program* berorientasi objek itu sudah berkembang secara pesat. Ini jelas mempermudah para praktisi pengembang sistem/program dalam melakukan aktifitasnya.

Daftar Pustaka

Harrington, Jan L., "*C++ and the Object Oriented Paradigm : as IS Perspective*", John Wiley & Son, Inc., New York, 2004.

Jogiyanto, "Sistem Teknologi Informasi Pendekatan Terintegrasi : Konsep Dasar, Teknologi, Aplikasi, Pengembangan, dan Pengelolaan", Edisi ke-2. Andi Offset, Yogyakarta, 2003.

Kadir, Abdul, "Pemrograman C++", Andi, Yogyakarta, 2004.

Mathiassen, Lars., Madsen, Andreas Munk, Nielsen, Peter Axel, dan Stage, Jan, "*Object Oriented Analysis & Design",* First Edition. Marko Publishing ApS, Aalborg, Denmark, 2000.

Pohl, Ira, "*Object Oriented Programming Using C++*", The Benjamin / Cummings Publishing Company, Inc., California, 2003.

Whitten, Jeffrey L., Bentley, Lonnie D., dan Dittman, Kevin C, "*System Analysis and Design Methods",* Fifth Edition. McGraw-Hill Companies, Inc., USA, 2001.