Memahami Konsep OOP dengan C++

Michael Lionardi
lionardi@web.de

Bab 1
Pembahasan Kelas Dalam C++

1.1 Pendahuluan

Tulisan ini merupakan pengenalan kepada pemrograman berorientasi objek (Object-oriented Programming, selanjutnya disebut OOP) dengan menggunakan ANSI C++. Disarankan agar Anda menguasai dasar-dasar pemrograman struktural terlebih dahulu dengan menggunakan salah satu bahasa pemrograman, baik C, Pascal, Basic atau yang lainnya. Sedikit sejarah tentang C++, C++ diciptakan oleh Bjarne Stroustrup di laboratorium Bell pada awal tahun 80-an, sebagai pengembangan dari bahasa C dan Simula. Saat ini, C++ merupakan salah satu bahasa yang paling populer untuk pengembangan software berbasis OOP. Tulisan ini memperkenalkan paradigma pemrograman berorientasi objek dengan menggunakan C++.

1.2 Bagaimana Konsep OOP?

Konsep utama pemrograman berorientasi objek yaitu melakukan permodelan objek dari kehidupan nyata ke dalam tipe data abstrak. Jelasnya, pemrograman berorientasi objek merupakan konsep pemrograman untuk memodelkan objek yang kita gunakan dalam kehidupan sehari-hari, dan konsep seperti ini membawa perubahan yang mendasar dalam konsep pemrograman terstruktur. Perubahan dramatis dalam konsep dasar disebut paradigma, maka jangan heran bila banyak orang yang menyebut “paradigma OOP” karena memang OOP membawa konsep yang sama sekali berbeda

dengan bahasa pemrograman generasi sebelumnya (bahasa pemrograman terstruktur). Setiap objek dalam kehidupan nyata dapat kita pandang sebagai kelas, misalnya kelas Hewan, kelas Manusia, kelas Mobil. Sedangkan objek dari kelas tersebut misalnya sapi dan ayam untuk kelas Hewan, Budi dan Tono untuk kelas Manusia serta Toyota dan VW untuk kelas Mobil. Dengan OOP, kita dapat mengimplementasikan objekt data yang tidak hanya memiliki ciri khas (attribut), melainkan juga memiliki metode untuk memanipulasi attribut tersebut. Singkatnya, OOP memiliki keunggulan dari konsep pemrograman terstruktur, selain itu juga memiliki kemampuan untuk mengimplementasikan objek dalam kehidupan nyata.

1.3 Struktur Kelas

Sebagai langkah pertama dalam OOP akan kita bahas pendefinisian kelas di C++. Dalam bagian 1.2 penulis telah mencontohkan beberapa kelas yang lazim kita temui dalam kehidupan sehari-hari. Mari kita amati contoh lain dari kehidupan kita, dengan mendeklarasikan sebuah kelas bernama *BilanganRasional* :

```
class BilanganRasional
{
public :
      void assign (int,int);
      void cetak();
private :
      int pembilang, penyebut;
};
```

Perhatikan contoh di atas. Untuk mendefinisikan sebuah kelas, dipakai kata kunci *class*, diikuti dengan pendeklarasian nama kelas tersebut. Fungsi assign() dan cetak() disebut *member function* (member fungsi). Sedangkan variabel pembilang dan penyebut disebut *member data* (member data atau member variabel). Disebut *member* karena kesemuanya merupakan anggota dari kelas *BilanganRasional*.

Perhatikan kata kunci *Public* dan *Private*. Member functions pada contoh di atas dideklarasikan sebagai fungsi global, sedangkan member data dideklarasikan sebagai lokal. Perbedaannya, member global dapat diakses dari luar kelas, sedangkan member lokal hanya dapat diakses dari kelas itu sendiri.

Sekarang, dimana kita telah menciptakan kelas *Bilangan Rasional*, kita dapat mendeklarasikan sebuah objek dari kelas *BilanganRasional* sebagai berikut :

```
      BilanganRasional objekBilangan;
```

Perhatikan bahwa disini *objekBilangan* merupakan nama dari objek tersebut, dan *BilanganRasional* merupakan nama kelas yang ingin kita buat objeknya. Proses pembuatan sebuah objek biasa disebut *penginstansian* (bukan penginstalasian), dan sebuah objek disebut *instans* (instance) dari sebuah kelas.

Untuk lebih jelasnya, perhatikan listing selengkapnya :

```
class BilanganRasional
{
public :
```

```
      void assign (int,int);
      void cetak();
private :
      int pembilang, penyebut;
};

void main()
{
      //mendeklarasikan objekBilangan seperti telah dibahas di atas
      BilanganRasional objekBilangan;

      // member fungsi assign() dipanggil.
      objekBilangan.assign (22,7);

      // member fungsi cetak() dipanggil.
      ObjekBilangan.cetak();
}
void BilanganRasional::assign(int pemb, int peny)
{
      pembilang = pemb;
      penyebut = peny;
}

void BilanganRasional::cetak()
{
      cout<<pembilang<<' / '<<penyebut;
}
```

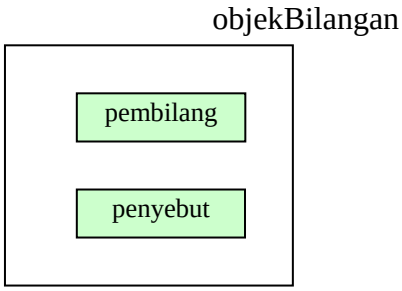
Perhatikan blok main(). Sekarang Anda sudah mempunyai sebuah objek bernama *objekBilangan* dari kelas *BilanganRasional*. Seperti Anda lihat, pendeklarasian sebuah objek sama seperti mendeklarasikan sebuah variabel. Atau dengan kata lain *objekBilangan* adalah sebuah objek dengan tipe *BilanganRasional*. Sekarang, bagaimana memanggil fungsi dari sebuah objek? Hal ini dapat dicapai dengan menghubungkan nama objek dan fungsi yang ingin dipanggil dengan operator tanda titik (.). Sehingga untuk memanggil fungsi *assign(),* dapat dilakukan dengan cara sebagai berikut :

```
objekBilangan.assign(22,7);
```

Nilai 22 dan 7 merupakan parameter yang diterima oleh fungsi *assign().* Di dalam fungsi tersebut, nilai 22 diinisialisasikan ke dalam member data *pembilang*, dan nilai 7 diinisialisasikan ke dalam member data *penyebut*. Sehingga bila fungsi cetak() dipanggil, maka akan diperoleh hasil sebagai berikut :

```
22 / 7
```

Sebagai tambahan perhatikan ilustrasi di bawah ini :



Gambar di atas merupakan ilustrasi dari objek *objekBilangan* dengan 2 member data, yakni *pembilang* dan *penyebut*.

Perhatikan juga bahwa semua pendeklarasian fungsi, baik fungsi assign() maupun fungsi cetak() didahului dengan penanda *BilanganRasional::* . Hal ini untuk menunjukkan kepada compiler agar compiler tidak “bingung”, untuk kelas mana fungsi tersebut dideklarasikan, karena di C++ biasanya sebuah fungsi diletakkan di file yang terpisah.

1.4 Konstruktor

Sebelumnya kita telah menggunakan member fungsi *assign()* untuk memasukkan nilai ke dalam member variabel *pembilang* dan *penyebut.* Sebuah konstruktor melakukan tugas yang sama dengan fungsi *assign(),* sehingga Anda tidak perlu repot-repot memanggil fungsi *assign()* untuk setiap objek yang Anda deklarasikan. Sebuah konstruktor harus mempunyai nama yang sama dengan kelas dimana konstruktor tersebut berada, dan dideklarasikan tanpa *return value* (nilai balik), juga tanpa kata kunci *void.* Mari kita kembangkan kelas *BilanganRasional* yang telah kita bahas sebagai berikut :

```
class BilanganRasional
{
public :
      //KONSTRUKTOR BilanganRasional
      BilanganRasional(int pemb, int peny)
      {
            pembilang = pemb;
            penyebut = peny;
      }
private :
            int pembilang, penyebut;
};
```

Bandingkan struktur konstruktor dengan fungsi *assign()* yang telah kita bahas sebelumnya. Konstruktor *BilanganRasional* melakukan tugas yang sama dengan member fungsi *assign()*. Bedanya hanya terletak pada pemanggilan fungsi dan konstruktor tersebut. Jika fungsi *assign()* harus kita panggil dengan didahului oleh pendeklarasian sebuah objek, kemudian fungsi dari objek tersebut dipanggil dengan operator titik disertai nilai yang ingin kita input, misal

```
BilanganRasional x;
x.assign(22,7);
```

maka konstruktor cukup dipanggil sebagai berikut :

```
BilanganRasional x(22,7);
```

Kedua varian tersebut melakukan hal yang sama, yakni menginitialisasikan nilai 22 ke member variabel *pembilang*, dan nilai 7 ke variabel *penyebut.*

1.5 Konstruktor Dengan Initialization Lists

Penulisan konstruktor dengan daftar initialisasi *(initialization lists)* merupakan fasilitas yang disediakan oleh C++ untuk menyederhanakan struktur konstruktor. Ini berarti, contoh konstruktor di atas dapat pula ditulis sebagai berikut :

```
class BilanganRasional
{
public :
      BilanganRasional(int pemb, int peny) : pembilang(pemb),
      penyebut(peny) { }

private :
      int pembilang, penyebut;
};
```

Contoh di atas menghasilkan fungsi yang sama dengan konstruktor yang kita bahas sebelumnya.

1.6 CopyConstructor

Sampai sejauh ini kita telah mempelajari bagaimana struktur sebuah konstruktor serta bagaimana membuat objek dari konstruktor yang telah didefinisikan. Akan tetapi, coba bayangkan apabila Anda telah mempunyai sebuah objek x, dan kemudian Anda menginginkan membuat sebuah objek y yang memiliki nilai member data dan member fungsi yang sama. Tentu saja Anda dapat mendeklarasikan objek baru dengan memanggil konstruktor yang sama sebanyak 2 kali :

```
BilanganRasional x(22,7);
BilanganRasional y(22,7);
```

Perintah di atas mendeklarasikan 2 objek, yakni x dan y yang masing-masing memiliki nilai 22 pada member variabel *pembilang* dan 7 pada member variabel *penyebut.* Akan tetapi, Anda dapat juga mempersingkat kode diatas dengan perintah berikut :

```
BilanganRasional x(22,7);
BilanganRasional y(x);
```

Berikut listing contoh untuk Copy Constructor :

```
class BilanganRasional
{
public :
      BilanganRasional(int pemb, int peny) : pembilang(pemb),
      penyebut(peny) { }

      //CopyConstructor terdapat disini
      BilanganRasional(const BilanganRasional& br) :
      pembilang(br.pembilang), penyebut(br.penyebut) { }

private :
      int pembilang, penyebut;
};

void main()
{
      BilanganRasional x(22,7);
      BilanganRasional y(x);
}
```

Deklarasi CopyConstructor otomatis dipanggil ketika Anda mengkopi objek x ke objek y. Perhatikan bahwa x menjadi parameter ketika kita mendeklarasikan objek y.

1.6 Destruktor

Jika kita mendeklarasikan konstruktor untuk membuat sebuah objek, maka kita juga harus mendeklarasikan sebuah *destruktor* untuk menghapus sebuah objek. Setiap kelas mempunyai tepat

satu destruktor. Jika Anda tidak mendeklarasikan sebuah destruktor dalam sebuah kelas, maka destruktor otomatis akan diciptakan sendiri oleh compiler C++. Destruktor dapat kita definisikan sendiri dengan simbol ~. Disarankan untuk mendefinisikan sendiri destruktor walaupun secara otomatis compiler C++ akan mendeklarasikan sebuah destruktor pada saat program Anda dicompile, tetapi dengan mendefinisikan sendiri sebuah destruktor maka Anda mempunyai kontrol penuh terhadap apa yang dilakukan destruktor dari kelas Anda. Perhatikan listing di bawah :

```
class BilanganRasional
{
public :
      BilanganRasional()  {cout <<"Konstruktor dipanggil\n";}

      //Destruktor dari kelas BilanganRasional
      ~BilanganRasional()  {cout <<"Destruktor dipanggil\n";}

private :
      int pembilang, penyebut;

};

void main()
{
      BilanganRasional x;
      cout<<"Disini main program\n" ;
}
```

Listing di atas akan menghasilkan output sebagai berikut :

```
Konstruktor dipanggil
Disini main program
Destruktor dipanggil
```

Dari contoh di atas dilihat bahwa konstruktor dipanggil ketika objek x dibuat. Sedangkan destruktor secara otomatis dipanggil oleh compiler ketika objek x meninggalkan blok main(). Hal ini sesuai dengan kaidah kelokalan objek di C++.

1.7 Ringkasan

Pada bab ini Anda telah mengetahui konsep OOP, mengapa OOP disebut paradigma serta apa bedanya konsep pemrograman berorientasi objek dengan konsep pemrograman terstruktur. Anda juga telah belajar mendefinisikan sebuah kelas, mendefinisikan member fungsi dan member data

serta struktur konstruktor dan destruktor dari sebuah kelas. C++ juga menyediakan fasilitas jika Anda ingin membuat duplikat sebuah objek, yaitu menggunakan fasilitas CopyConstructor. Selain itu Anda juga telah belajar mendeklarasikan sebuah konstruktor dengan Initialization Lists, sehingga pendeklarasian konstruktor Anda menjadi lebih efisien.

Daftar Pustaka :

1. Erlenkötter, Helmut. C++, Objektorientiertes Programmieren von Anfang an
2. http://infocom.cqu.edu.au/Staff/Mike_Turnbull/Home_Page/Lecturetts/Sect101.htm
3. http://www.aw-bc.com/catalog/academic/product/0,4096,0201895501-PRE,00.html
4. http://www.desy.de/gna/html/cc/Tutorial/node3.htm
5. http://www.uni-koeln.de/rrzk/kurse/unterlagen/CPPKURS/
6. http://www.uni-koeln.de/rrzk/kurse/unterlagen/CPPKURS/HTML/begriffe.htm
7. Hubbard, John R. Ph.D. Schaum’s Outlines. Programming with C++.