Konsep tambahan paradigma objek (1)

oleh Satrio Adi Rukmono

Pada Bab 1 telah dibahas bahwa konsep dasar paradigma objek hanya terdiri atas tiga hal: *messaging*, *isolation*, dan *extreme late-binding*. Meskipun demikian, lanskap pemrograman berorientasi objek hingga saat ini telah berevolusi hingga memiliki konsep-konsep tambahan yang menjadi intrinsik dalam bahasa dan sistem pemrograman berorientasi objek yang umum digunakan. Bab ini membahas masing-masing konsep tersebut secara singkat; setiap konsep ini berikutnya dibahas lebih mendalam dalam babnya masing-masing.

Objek

David West menyebut objek sebagai partikel terkecil yang membentuk alam semesta. Hal ini memiliki makna bahwa semua yang ada di "semesta" perangkat lunak tersusun atas objek-objek. Objek memiliki perilaku tertentu untuk memenuhi suatu tanggung jawab yang disepakati, atau "layanan" yang tersedia. Objek harus memiliki akses terhadap informasi yang dibutuhkan untuk menjalankan tanggung jawabnya. Informasi tersebut bisa dimiliki sendiri ataupun ditanyakan ke objek lain. Sebagai analogi, contohnya ketika ada yang meminta nomor KTP (NIK), kita akan "menanyakan" dulu NIK kita ke objek KTP yang ada di dalam dompet kita, kemudian "meneruskan" informasi tersebut ke yang meminta.

Objek didefinisikan oleh tanggung jawabnya, dengan kata lain suatu objek adalah enkapsulasi sekumpulan tanggung jawab. Sebuah objek mematuhi suatu antarmuka, yang menunjukkan hal-hal apa saja yang dapat dilakukan oleh objek tersebut (dengan kata lain, apa saja tanggung jawab objek tersebut). Objek tersusun atas objek-objek lainnya dalam hubungan *has-a* ("memiliki sebuah") yang dapat dicontohkan dengan objek mobil yang tersusun atas sebuah objek mesin ("mobil *memiliki sebuah* mesin"), empat buah objek roda, dan objek-objek lainnya. Objek yang menyusun objek lainnya disebut sebagai **atribut** bagi objek lain tersebut, sehingga dalam contoh sebelumnya mesin merupakan salah satu atribut dari mobil. Atribut (disebut juga dengan *instance variable*, *field*, dan lain-lain*) tidak diekspos ke dunia luar. Masih dengan analogi mobil, sebagai pengemudi kita tidak langsung berinteraksi dengan mesin, melainkan dengan sekumpulan tombol, tuas, pedal, dan *antarmuka-antarmuka* lainnya yang dimiliki mobil, yang "di balik layar" sebenarnya melibatkan mesin untuk memenuhi tanggung jawabnya.

*) Kosakata C++ menyebut atribut sebagai *data member*, namun istilah ini mengerdilkan makna objek menjadi seolah hanya wadah untuk menyimpan data (dan fungsi).

Contoh tanggung jawab dan antarmuka objek "mobil"

Tanggung jawab	Antarmuka
Menambah kecepatan	Pedal gas
Mengurangi kecepatan	Pedal rem
Berganti arah/berbelok	Setir
...	...

Selain atribut, objek juga memiliki **metode**, yaitu deretan instruksi yang akan dieksekusi oleh sebuah objek ketika menerima pesan yang berkorespondensi dengan metode tersebut. Dalam kosakata C++ metode disebut juga dengan *function member* dan didefinisikan sebagaimana sebuah fungsi:

```
/* Metode push dimiliki semua objek bertipe Stack */
void Stack::push(element_type e) {
  if (!is_full()) {
    internal_buffer[top++] = e;
  }
}
```

Secara umum terdapat dua kelompok metode, yaitu metode yang mengembalikan sebuah objek dan metode yang mengubah status dari objek yang memilikinya.

Jika objek memiliki *atribut* (analog dengan *data*) dan *method* (analog dengan *prosedur/fungsi*), apa bedanya dari *abstract data type* (ADT)? Sebuah objek tidak mengekspos "isi perutnya" ke objek lain—objek lain tidak boleh tahu bagaimana sebuah objek mengelola informasi yang dimilikinya secara internal. Secara konseptual, method merefleksikan ekspektasi pada domain persoalan, sedangkan fungsi merefleksikan detail implementasi pada program. Berbeda dari ADT, perancangan objek harus dimulai dari tanggung jawab (method) apa saja yang dimiliki suatu objek, dilanjutkan dengan memutuskan informasi (atribut) apa saja yang diperlukan objek untuk menjalankan tanggung jawab tersebut.

Kelas dan prototipe

Kebanyakan bahasa pemrograman berorientasi objek menggunakan **kelas** untuk mendeskripsikan sekelompok objek dengan tipe yang sama. Kelas mendefinisikan atribut dan metode apa saja yang dimiliki semua objek dari kelas tersebut. Dalam

konteks ini, sebuah objek dikatakan sebagai sebuah *instance* dari sebuah kelas. Analogi yang sering digunakan untuk kelas adalah *cetak biru* (*blueprint*): kita hanya memerlukan satu cetak biru untuk sebuah jenis mobil; dari cetak biru tersebut kita bisa membuat banyak mobil dari jenis yang sama. Meskipun misalnya sebuah mobil berwarna merah (atribut "warna" bernilai "merah") dan mobil lainnya berwarna biru, jika keduanya dibuat berdasarkan cetak biru yang sama maka keduanya merupakan *instance* yang berbeda dari sebuah kelas yang sama.

Dalam sistem objek murni, semua adalah objek, termasuk kelas. Kita dapat mengirim pesan tertentu kepada sebuah kelas dan ia akan mengembalikan sebuah objek yang sesuai dengan deskripsi yang dimilikinya. Sebuah kelas memiliki metode khusus untuk menciptakan objek sesuai deskripsi tersebut; metode seperti ini disebut dengan **konstruktor**. Pada sudut pandang ini kelas dapat dianggap sebagai sebuah objek pabrik yang menciptakan objek-objek.

```
// java
aStack = new Stack(); // pemanggilan konstruktor kelas Stack
                      // dengan kata kunci `new`.

# ruby
aStack = Stack.new() # sintaks pemanggilan konstruktor tidak ada
                     # bedanya dari pengiriman pesan `new` kepada
                     # objek `Stack`.
```

Alternatif dari konsep kelas adalah konsep **prototipe**. Pada sistem objek berbasis prototipe, pemrogram mendefinisikan objek dengan atribut-atribut bernilai *default*. Objek ini disebut dengan prototipe. Untuk membuat sebuah *instance* yang unik, dibuat duplikat dari prototipe kemudian nilai atribut dari duplikat ini diubah sesuai kebutuhan. Sebagai contoh, mungkin prototipe sebuah mobil memiliki warna hitam; untuk membuat mobil berwarna biru, buat dulu duplikat mobil hitam kemudian ubah warnanya menjadi biru.

Pada praktiknya, ketika dihadapkan dengan bahasa pemrograman berbasis prototipe, banyak pemrogram mencoba meniru perilaku kelas. Dengan demikian dapat disimpulkan bahwa sejauh ini sistem berbasis kelas masih "menang" dari sistem berbasis prototipe. Hal ini bisa saja berubah di masa depan, mengingat pemrograman merupakan bidang yang selalu berevolusi.