Bagaiamana Berfikir Seperti Imuwan Komputer

Allen Downey
Diterjemahkan oleh Tim Buku Prodase - Universitas Telkom

Daftar Isi

Daftar Tabel

Daftar Gambar

Bab 1

Cara Program

Tujuan dari buku ini adalah mengajarkan kepada kamu bagaimana berfikir seperti seorang ilmuan komputer. Saya sangat suka bagaimana seorang ilmuan komputer berfikir karena mereka menggabungkan aspek-aspek terbaik dari ilmu Matematika, ilmu Teknik, dan ilmu pengetahuan alam. Sebagaimana halnya Matematikawan, ilmuan komputer menggunakan bahasa formal untuk menyatakan ide khususnya yang berkenaan dengan komputasi. Seperti Insinyur(*Engineer*), ilmuan komputer juga merancang sesuatu, merangkai beberapa komponen kedalam sebuah sistem dan mengeveluasi kelebihan dan kekurangan dari berbagai alternatif solusi. Mirip dengan ilmuan pada umumnya, para ilmuan komputer melakukan observasi tingkah laku dari sistem yang kompleks, membentuk beberap hipotesis dan menguji prediksi yang mereka buat.

Satu-satunya keahlian yang paling penting bagi seorang ilmuan komputer adalah keahlian dalam memecahkan masalah (*problem-solving*). Yang saya maksudkan dengan kemampuan memecahkan masalah adalah kemampuan mereka dalam melakukan formulasi masalah, berfikir secara kreatif mengenai solusi dari masalah yang telah diformulasikan dan mengekspresikan sebuah solusi secara jelas dan akurat. Dan ternyata

proses yang kamu lalui ketika belajar program komputer adalah sebuah kesempatan yang istimewa dalam berlatih keahlian memecahkan masalah. Itu kenapa judul dari bab ini adalah "Cara Program".

Pada satu sisi kamu akan belajar pemrograman, yang mana merupkah keahlian yang sangat penting. Disisi lainya, kamu akan menggunakan pemrograman sebagai sarana untuk belajar

1.1 Apa itu bahasa pemrograman?

Bahasa pemrograman yang akan kamu **p**elajari adalah Java, yang termasuk sebuah bahasa pemrograman yang relatif baru (Dirilis pertama kali oleh Sun Microsystem pada may 1995). Java adalah salah satu contoh dari bahasa pemrograman level tinggi (*high-level language*); bahasa pemrograman lain yang juga termasuk kategori bahasa pemrograman level tinggi adalah bahasa Python, C atau C++ dan Perl.

Selain istilah bahasa pemrograman level tinggi, terdapat juga istilah bahasa pemrograman level rendah (*low level languages*) dan terkadang dikenal juga dengan istilah bahasa mesin atau bahasa *assembly*. Pada kenyataanya, komputer hanya bisa memahami bahasa pemrograman level rendah. Oleh sebab itu, sebuah program yang ditulis menggunakan bahasa level tinggi harus diterjemahkan terlebih dahulu ke bentuk bahasa level rendah sebelum program tersebut dijalankan. Proses penterjemahan ini membutuhkan waktu sebelum bisa dijalankan oleh komputer, hal ini menjadi salah satu kekurangan dari bahasa pemrograman level tinggi.

Keunggulan dari bahasa level-tinggi cukup banyak jika dibandingkan dengan kekuranganna. Pertama, jauh lebih mudah untuk membuat program dengan menggunakan bahasa level-tinggi; waktu yang dibutuhkan untuk menuliskan program jauh lebih singkat, penulisannya juga jauh lebih pendek dan mudah dibaca jika dibandingkan dengan bahasa level-rendah. Keuntungan yang kedua adalah portabilitas dalam

menjalankannya diberbagai macam arsitektur komputer dengan tanpa modifikasi. Berbeda halnya dengan program yang ditulis dengan bahasa level-rendah yang hanya bisa dijalankan di komputer tertentu, sehingga perlu dimodifikasi jika ingin dijalankan pada komputer dengan arsitektur yang berbeda.

Oleh karena kelebiha-kelebihan tersebut, maka hampir semua program ditulis dengan menggunakan bahasa pemrograman level-tinggi. Bahasa level-rendah hanya digunakan untuk membuat program-program tertentu saja yang jumlahnya juga sedikit.

Terdapat dua cara untuk menterjemahkan sebuah program; **interpretasi** (*interpreting*) dan **kompilasi** (*compiling*). Sebuah *interpreter* adalah sebuah program yang membaca sebuah program level-tinggi dan melakukan apa yang diminta oleh program tersebut. Sebagai akibatnya, interpreter akan menterjemahkan program baris demi baris Sementara *complier* adalah sebuah program yang membaca sebuah program level-tinggi dan menterjemahkan keseluruhan program secara langsung, sebelum menjalankan perintah apa pun dari program tersebut. Sering kali, kamu akan melakukan proses kompilasi secara terpisah terlebih dahulu, kemudian baru menjalankan program. Pada kasus ini, program level-tinggi disebut dengan istilah kode sumber (*source code*) dan program yang telah diterjemahkan disebut dengan istilah kode objek (*object code*) atau *executable*.

Bab 2

Pengkondisian dan Rekursi

2.1 Operator Modulus

Operator modulus bekerja pada bilangan bulat (dan ekspresi integer) dimana hasilnya adalah hasil dari pembagian antara operan (bilangan) pertama dan operan kedua. Dalam Java, simbol dari operator modulus adalah tanda persen (%). Untuk sintaksnya sama seperti operator lainnya :

1. int hasilBagi = 7 / 3;
2. int sisaBagi = 7 % 3;

Operator pertama adalah pembagian bilangan bulat dengan hasil 2. Sedangkan operator kedua adalah operator modulus yang menghasilkan 7 sisa bagi dari 2 yaitu 1

Operator modulus tenyata sangat mengejutkan dan sangat berguna. Misalkan, kamu dapat memeriksa apakah suatu angka akan habis jika dibagi angka lain: jika x % y adalah nol, maka x habis dibagi y.

Dan juga anda dapat menggunakan operator modulus untuk mengekstrak digit paling kanan dari atau digit sebuah nomor. Misalnya, x % 100 menghasilkan digit paling kanan dari x (dalam basis 10). Demikian pula x % 100 akan menghasilkan dua digit terakhir dari x

2.2 Eksekusi Pengkondisian

Untuk menulis program yang berguna, kita hampir selalu perlu untuk memeriksa kondisi dan mengubah perilaku program yang sesuai. Pernyataan bersyarat memberikan kita kemampuan untuk melakukan ini. Bentuk sederhananya adalah jika pernyataan :

```
if (x >0) {
System.out.println("x adalah bilangan positif");
}
```

Ekspresi dalam kurung "if (x >0)" disebut kondisi. Jika benar, maka pernyataan dalam kurung dijalankan. Jika kondisi ini salah, tidak ada yang terjadi.

Kondisi dapat berisi banyak operator perbandingan, terkadang disebut **Operator Relasional**

```
x == y // x sama dengan y
x != y // x tidak sama dengan y
x >y // x lebih besar daripada y
x <y // x is lebih kecil daripada y
x >= y // x lebih besar atau sama dengan y
x <= y // x lebih kecil atau sama dengan y
```

Meskipun operasi ini familiar, sintaks pada java menggunakan simbol matematika yang sedikit berbeda seperti =, 6= dan <=. Kesalahan

yang sering dilakukan adalah menggunakan satu sama dengan (=) untuk membandingkan nilai bukan menggunakan dobel sama dengan (==). Ingat bahwa = adalah operator penugasan, dan == adalah operator perbandingan.

Kedua operan yang dibandingkan harus memiliki tipe data yang sama. Misal, anda hanya bisa membandingkan int ke int dan double ke double.
Operator == dan != juga bekerja pada tipe data String, tetapi mereka tidak melakukan seperti apa yang anda ekspektasikan. Dan operator relasional yang lain secara keseluruhan tidak melakukan apa apa. Kita akan mempelajari bagaimana cara membandingkan string pada **Bagian 8.10.**

2.3 Eksekusi Alternatif

Bentuk kedua dari eksekusi pengkondisian adalah eksekusi alternatif, dimana ada dua kemungkinan, dan kondisi menentukan bagian mana yang akan dieksekusi. Sintaksnya seperti :

```
if (x%2 == 0) {
System.out.println("x adalah bilangan genap");
} else {
System.out.println("x adalah bilangan genap");
}
```

Jika sisa bagi dari x dibagi dengan 2 adalah 0, maka kita tahu bahwa x adalah bilangan genap, dan kode ini berguna untuk mencetak pernyataan tersebut. Jika kondisi salah maka peryataan kedua yang dicetak yaitu x adalah bilangan ganjil. Karena kondisi harus benar atau salah, maka hanya tepat satu alternatif yang akan dieksekusi.
Sebagai sampingan jika anda sering memeriksa paritas (pemerataan

atau keanehan) nomor, maka anda dapat ”membungkus” kode ini ke dalam sebuah method, sebagai berikut:

```
public static void printParity(int x) {
if (x%2 == 0) {
System.out.println(”x adalah bilangan genap”);
} else {
System.out.println(”x adalah bilangan genap”);
}
}
```

Sekarang anda memiliki method yang bernama printParity yang berfungsi untuk mencetak pesan yang sesuai untuk setiap bilangan bulat yang anda berikan. Dalam method utama, anda dapat memanggil method printParity dengan cara :

```
printParity(17);
```

Selalu ingat ketika anda memanggil sebuah method, tidak perlu mendeklarasikan argumen. Karena java dapat mengetahui jenis tipe data mereka. Dan anda juga harus dapat menahan godaan untuk menulis hal-hal seperti :

```
int number = 17;
printParity(int number); // WRONG!!!
```

2.4 Kondisi Berantai

Terkadang anda ingin memeriksa sejumlah kondisi terkait dan memilih salah satu dari beberapa tindakan. Salah satu cara untuk melakukan ini adalah dengan merantaikan antara if dan elsenya:

```
if (x >0) {
System.out.println(”x adalah bilangan positif”);
} else if (x <0){
```

```
System.out.println("x adalah negatif positif");
} else { System.out.println("x adalah nol"); }
```

Rantai ini bisa sepanjang yang anda inginkan, meskipun bisa sulit untuk dibaca. Salah satu cara untuk membuat mereka lebih mudah dibaca adalah dengan menggunakan penulisan standar dengan lekukan, seperti yang ditunjukkan dalam contoh ini. Jika Anda menyimpan semua pernyataan dengan penulisan standar, maka akan meminimalisir kesalahan penulisan sintaks dan error serta lebih mudah untuk menemukan mereka jika anda melakukannya.

2.5 Kondisi Bersarang

Selain dirantaikan, kondisi juga dapat disarangkan antara satu kondisi dengan kondisi yang lain. Kita dapat menuliskan contoh dari kondisi bersarang sebagai berikut:

```
if (x >0) {
System.out.println("x adalah bilangan positif");
} else { if(x <0) {
System.out.println("x adalah negatif positif");
} else { System.out.println("x adalah nol"); } }
```

Sekarang terdapat terdapat kondisi dengan dua cabang, Cabang yang pertama berisi perintah untuk mencetak pernyataan sederhana, tetapi cabang yang kedua berisi pernyataan kondisi yang lain yang memiliki dua cabang. Kedua cabang kondisi tersebut sama sama untuk mencetak pernyataan sederhana. Tetapi mereka adalah pernyataan kondisi yang benar.
Lekukan membantu membuat struktur yang jelas, namun demikian, kondisi bersarang sulit jika dibaca dengan sangat cepat. Maka hindari mereka sebisa mungkin.

Di sisi lain, struktur bersarang semacam ini adalah hal yang umum, dan kita pasti akan menemuinya lagi, sehingga anda dapat lebih baik untuk menggunakannya.

2.6 Pernyataan Pengembalian

Pernyataan pengembalian memungkinkan anda untuk mengakhiri eksekusi method sebelum Anda mencapai akhir. Salah satu alasan untuk menggunakannya adalah jika Anda mendeteksi kesalahan kondisi:

```
public static void printLogarithm(double x) {
if (x <= 0.0) {
System.out.println("Tolong masukkan bilangan positif saja");
return;
}
double result = Math.log(x);
System.out.println("Hasil log dari x adalah " + result);
}
```

Baris kode diatas mendefinisikan sebuah method printLogarithm dengan parameter berupa tipe data double yang bernama x. Method ini berfungsi untuk memeriksa apakah x kurang dari atau sama dengan nol, jika benar maka method ini akan mencetak pesan error dan menggunakan return untuk keluar dari method. Aliran dari eksekusi, segera mengembalikan segera kembali ke pemanggil dan sisa baris di method tidak dieksekusi.

2.7 Tipe Konversi

Anda mungkin bertanya-tanya bagaimana ekspresi seperti **"Hasil dari log x adalah" + result** tidak error. Hal ini karena salah satu operan bertipe **String** dan yang lainnya bertipe **double**. Inilah salah satu keunggulan dari Java yaitu, secara otomatis mengkonversi operan yang bertipe double menjadi String sebelum ia melakukan penggabungan antar string.
Setiap kali Anda mencoba untuk "menambahkan" dua ekspresi, jika salah satu dari mereka adalah **String**, Java mengubah String lain, kemudian melakukan penggabungan string (concatination). Menurut anda apa yang akan terjadi jika anda melakukan operasi antara sebuah integer dan sebuah nilai floating-point?

2.8 Rekursi

Seperti yang saya sebutkan dalam bab terakhir bahwa salah satu method untuk boleh memanggil method yang lain, dan kita juga telah melihat beberapa contohnya. Saya tidak menyebutkan bahwa sebuah metode boleh untuk memanggil dirinya sendiri. Ini mungkin tidak jelas mengapa itu adalah hal yang baik, tapi ternyata menjadi salah satu hal yang paling ajaib dan menarik yang dapat dilakukan sebuah program .
Sebagai contoh, lihat method berikut:

```
public static void countdown(int n) {
if (n == 0) {
System.out.println("Waktu Habis!");
} else {
System.out.println(n);
countdown(n-1);
}
```

```
}
```

Nama method ini adalah countdown dan dibutuhkan satu bilangan bulat sebagai parameter. Jika parameternya nol, maka mencetak kata "Waktu Habis!" Jika tidak, akan mencetak nomor dan kemudian memanggil method countdown sendiri dengan **n - 1** sebagai argumen. Apa yang terjadi jika kita memanggil metode ini pada method utama seperti ini:

```
countdown (3);
```

Eksekusi method countdown akan dimulai dengan n = 3, karena n tidak nol, maka akan mencetak nilai 3, dan kemudian memanggil method countdown kembali...

Eksekusi method countdown akan dimulai dengan n = 2, karena n tidak nol, maka akan mencetak nilai 2, dan kemudian memanggil method countdown kembali...

Eksekusi method countdown akan dimulai dengan n = 1, karena n tidak nol, maka akan mencetak nilai 1, dan kemudian memanggil method countdown kembali...

Eksekusi method countdown akan dimulai dengan n = 0, karena n adalah nol, maka akan mencetak "Waktu Habis!", dan mengembalikan nilai **n**...

Method countdown ini menghasilkan **n=1**

Method countdown ini menghasilkan **n=2**

Method countdown ini menghasilkan **n=3**

Kemudian anda kembali pada method utama, maka output secara keseluruhannya adalah :
3
2
1
Waktu Habis!
Sebagai contoh kedua, mari kita lihat lagi di method newLine dan threeline.

```
public static void newLine() {
System.out.println ("");
}

public static void threeLine() {
newLine();
newLine();
newLine();
}
```

Pekerjaan ini tidak akan banyak membantu jika kita ingin mencetak 2 atau 106 baris. Sebuah alternatif yang lebih baik adalah:

```
public static void nLines(int n) {
if (n <0) {
System.out.println ("");
nLines (n-1);
}
}
```

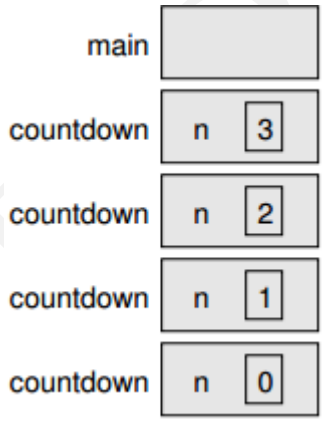
Program ini mirip dengan countdown; selama n lebih besar dari nol, ia akan mencetak baris baru dan kemudian memanggil dirinya dengan argumen **n-1** untuk mencetak baris tambahan. Jumlah dari baris yang bisa dicetak adalah 1 + (n-1).

Ketika metode memanggil dirinya sendiri disebut **rekursi**, dan metodenya adalah **rekursif**

2.9 Diagram Stack untuk Method Rekursif

Dalam bab sebelumnya kita menggunakan diagram stack untuk mewakili negara dari sebuah program selama pemanggilan method. Diagram yang sama dapat mempermudah untuk menafsirkan metode rekursif. Ingat bahwa setiap kali sebuah method dipanggil itu menciptakan kerangka baru yang berisi versi baru dari parameter method dan variabel.

Gambar berikut adalah diagram stack untuk method countdown, dengan n = 3:



Ada satu frame untuk utama dan empat frame untuk method countdown, masing-masing dengan nilai yang berbeda untuk parameter n. Pada bagian paling bawah tumpukan, terdapat method countdown

dengan n = 0, tumpukan ini disebut **base case**. Hal ini tidak membuat panggilan rekursif, sehingga tidak ada lagi frame untuk menjalankan method countdown. Frame untuk utama adalah kosong karena utama tidak memiliki parameter atau variabel.

2.10 Rangkuman

Modulus: Sebuah operator yang bekerja pada bilangan bulat dan menghasilkan sisa bagi ketika satu nomor dibagi dengan yang lain. Di java modulus dilambangkan dengan tanda persen (%).

Kondisional: Sebuah blok pernyataan yang mungkin atau mungkin tidak akan dieksekusi tergantung pada beberapa kondisi.

Chaining: Sebuah cara untuk menggabungkan beberapa pernyataan kondisional secara berurutan.

Nesting: Menempatkan pernyataan kondisional dalam salah satu atau kedua cabang pernyataan kondisional lain.

Typecast: Sebuah operator yang mengkonversi dari satu jenis tipe data ke tipe data yang lain. Di java biasanya nama tipe data berada dalam tanda kurung, seperti (int).

Rekursi: Proses memanggil metode yang sama yang sedang anda eksekusi.

Base Case: Sebuah kondisi yang menyebabkan metode rekursif tidak untuk membuat panggilan rekursif.

2.11 Latihan

4.1. Gambarkan diagram stack yang menunjukkan keadaan dari program dalam Bagian 4.8 setelah method utama memanggil nLines dengan parameter n = 4, tepat sebelum pemanggilan terakhir nLines dikembalikan.

4.2. Latihan ini adalah meninjau aliran eksekusi melalui program dengan beberapa metode. Baca kode berikut dan menjawab pertanyaan-pertanyaan di bawah ini.

```
    public class Buzz {
    public static void baffle(String blimp) {
    System.out.println(blimp);
    zippo("ping", -5);
    }

    public static void zippo(String quince, int flag){
    if (flag <0) {
    System.out.println(quince + " zoop");
    } else {
    System.out.println("ik");
    baffle(quince);
    System.out.println("boo-wa-ha-ha");
    }
    }

public static void main(String[] args) {
    zippo("rattle", 13);
    }
```

}

1. Tuliskan nomor 1 di sebelah pernyataan pertama dari program ini yang akan dieksekusi. Hati-hati untuk membedakan hal-hal yang pernyataan dari hal-hal yang tidak. 2. Tuliskan nomor 2 di sebelah pernyataan kedua, dan seterusnya sampai akhir program. Jika pernyataan dieksekusi lebih dari sekali, itu mungkin berakhir dengan lebih dari satu nomor sebelahnya. 3. Berapa nilai dari parameter **blimp** saat **baffle** akan dipanggil? 4. Apa output dari program ini?

Latihan 4.3. Lirik pertama dari lagu "99 Botol Bir" adalah:

99 bottles of beer on the wall, 99 bottles of beer, ya take one down, ya pass it around, 98 bottles of beer on the wall.

Lirik berikutnya adalah identik kecuali bahwa jumlah botol semakin kecil oleh salah satu di setiap lirik, sampai lirik terakhir:

No bottles of beer on the wall, no bottles of beer, ya cant take one down, ya cant pass it around, cause there are no more bottles of beer on the wall!

Dan kemudian lagu (akhirnya) berakhir.

Tulis sebuah program yang mencetak seluruh lirik "99 Botol bir." Program harus mencakup metode rekursif yang melakukan bagian yang sulit, tapi anda mungkin ingin menulis method tambahan untuk memisahkan fungsi utama program.

Ketika Anda mengembangkan kode Anda, uji dengan sejumlah lirik kecil, seperti "3 Botol Bir. "

Tujuan dari latihan ini adalah untuk mengambil masalah dan memecahnya menjadi lebih kecil masalah, dan untuk memecahkan masalah yang lebih kecil dengan menulis metode sederhana.

Latihan 4.4. Apa output dari program ini?

```
public class Narf {
public static void zoop(String fred, int bob) {
System.out.println(fred);
if (bob == 5) {
ping("not ");
} else {
System.out.println("!");
}
}

public static void main(String[] args) {
int bizz = 5;
int buzz = 2;
zoop("just for", bizz);
clink(2*buzz);
}

public static void clink(int fork) {
System.out.print("It's ");
zoop("breakfast ", fork) ;
}

public static void ping(String strangStrung) {
System.out.println("any " + strangStrung + "more ");
}
```

}

Latihan 4.5. Fermat terakhir Teorema mengatakan bahwa tidak ada bilangan bulat a, b, dan c sehingga:
an + bn = cn

kecuali dalam kasus ketika n = 2.

Tulis sebuah metode checkFermat bernama yang mengambil empat bilangan bulat sebagai parameter a, b, c dan n. Lalu periksa untuk melihat apakah teorema Fermat benar. Jika n lebih besar dari 2 dan ternyata benar bahwa an + bn = cn, dan program harus mencetak "Teori Fermat adalah salah!" Jika tidak program harus mencetak "Tidak, yang tidak bekerja. "
Anda harus mengasumsikan bahwa ada method raiseToPow bernama yang mengambil dua bilangan bulat sebagai argumen dan memunculkan argumen pertama untuk kekuatan kedua. Sebagai contoh:

int x = raiseToPow (2, 3);
akan menetapkan nilai 8 untuk x, karena $2^3 = 8$.