# Branching

Pada minggu ini kita akan mempelajari mengenai ekspresi *boolean* serta *branching*. Untuk dapat memecahkan permasalahan dengan *branching*, Anda dituntut untuk jeli dalam menangkap setiap kondisi yang mungkin terjadi. Pastikan Anda memahami permasalahannya terlebih dahulu serta merancang ide *problem solving*-nya sebelum anda mulai menuliskan kode program!

## 1. Ekspresi *Boolean*

Pada bagian ini, kita akan berlatih menggunakan ekspresi *boolean* pada Java. Untuk memudahkan demonstrasi, anda perlu membuat file dengan nama `DemoBoolean` sebagai tempat anda mencoba kode program.

```
public class DemoBoolean
{
    public static void main(String args[])
    {
        char demoChar='q';

        boolean demoVar1;
        demoVar1=demoChar=='q';
        System.out.println(demoVar1);

    }
}
```

*Gambar 1 Program DemoBoolean*

Perhatikan program DemoBoolean pada Gambar 1! Apa arti dari ekspresi `demoVar1=demoChar=='q'`? Tambahkanlah perintah-perintah untuk melakukan hal berikut ini:

- Buat sebuah variabel dengan nama `demoVar2` dan isi nilainya dengan ekspresi yang memeriksa apakah isi demoChar tidak sama dengan huruf `'a'`. *Print* isi variabel tersebut ke layar.

Tambahkanlah kode untuk melakukan hal-hal di bawah ini dengan menggunakan ekspresi logika! Untuk mengerjakan bagian ini, anda perlu menambahkan dua buah variabel lokal dengan nama `demoInt1` dan `demoInt2`. Isi nilai variabel `demoInt1` dengan 5 dan `demoInt2` dengan 12. Tambahkan variabel-variabel lokal `demoVar` bertipe *boolean* yang dibutuhkan!

Catatan: Seluruh atribut dari `demoVar4` sampai dengan `demoVar11` **tidak** membutuhkan perintah if!

| **Buat ekspresi logika untuk ...** | **Dan simpan hasil ekspresi tersebut di...** |
|---|---|
| Memeriksa apakah `demoInt1` lebih besar atau sama dengan 0 | `demoVar4` |

| | |
|---|---|
| Memeriksa apakah `demoInt1` lebih kecil dari `demoInt2` | `demoVar5` |
| Memeriksa apakah `demoInt1*2` lebih besar dari `demoInt2` | `demoVar6` |
| Memeriksa apakah `demoInt1` lebih besar dari 0 **dan** lebih kecil dari 3 | `demoVar7` |
| Memeriksa apakah `demoInt1` lebih kecil dari 0 **atau** lebih besar dari 3 | `demoVar8` |
| Memeriksa apakah `demoInt1` adalah bilangan genap. (Hint: Anda perlu menggunakan operator modulo). | `demoVar9` |
| Memeriksa apakah `demoInt2` adalah bilangan ganjil | `demoVar10` |
| Memeriksa apakah `demoVar10` bernilai *false* | `demoVar11` |

Kumpulkan program ini pada tugas M0301XXYYY.java!

## 2. Branching

Dalam kehidupan sehari-hari, kita mengenal kata "jika" untuk menyatakan syarat dari sebuah keadaan (jika saya rajin belajar Daspro, seharusnya saya lulus kuliah tsb, dll). Demikian juga dengan bahasa pemrograman. Dengan *conditional control structure*, komputer dapat memilih antara dua atau lebih alternatif yang ada.

### a. If,if-else, block statement

Struktur *branching* dengan sebuah kondisi dapat dituliskan seperti pada Gambar 2 sebagai berikut:

```
if(...kondisi...){
    ...aksi...
}
```

*Gambar 2 Struktur Branching*

Sebuah struktur branching dapat mengandung *block statement*. *Block statement* ditandai dengan penggunaan sepasang tanda kurung kurawal ( '{' dan '}' ). Awal dari *block* ditandai dengan '{', sedangkan akhir dari *block* ditandai dengan '}'.

```
if(...kondisi... ){
    ...statement1...
    ...statement2...
    ...
    ...statementN...
}
```

*Gambar 3 Struktur Branching dengan block statement*

Gambar 3 menunjukkan sebuah *block statement* yang terdiri dari $N$ buah *statement* (`statement1`, …, `statementN`). Arti dari struktur tersebut adalah "jika kondisi yang tertera terpenuhi, maka $N$ buah *statement* tersebut akan dieksekusi."

Struktur *branching* dengan 2 kondisi dapat dituliskan seperti pada Gambar 4.

```
if(...kondisi...){
    ...aksi1...
}else{
    ...aksi2...
}
```

*Gambar 4 Struktur Branching if-else dengan block statement*

## Magic Spell (M0302XXYYY.java)

Harry Wotter adalah seekor Wombat yang sedang belajar sihir. Guru sihir Harry telah mengajarkan bahwa kalimat yang bersifat *magical* adalah kalimat yang huruf pertama dan huruf terakhirnya sama atau jarak perbedaaannya hanya satu. Sebagai contoh, berikut ini adalah contoh kalimat-kalimat yang bersifat *magical*:

- abra kadabra: huruf pertama (a) dan huruf terakhirnya (a) sama
- lab acadak: huruf pertama (l) dan huruf terakhirnya (k) berjarak 1
- bac: huruf pertama (b) dan huruf terakhirnya (c) berjarak 1

Buatlah sebuah program yang menerima masukan berupa sebuah kalimat dan akan menentukan apakah kalimat tersebut bersifat *magical* atau tidak!

### Spesifikasi Masukan

Masukan program ini adalah sebuah kalimat. Sebuah kalimat hanya dapat mengandung huruf non kapital. Sebuah kalimat dapat mengandung spasi.

### Spesifikasi Keluaran

Keluarkan "magical" bila kalimat tersebut bersifat magical dan "tidak magical" bila sebaliknya.

### Contoh Masukan dan Keluaran

| No. | Masukan | Keluaran Yang Diharapkan |
|---|---|---|
| 1 | `abra kadabra` | `magical` |
| 2 | `lab acadak acadak acadak` | `magical` |
| 3 | `bac bab` | `magical` |
| 4 | `bad zzz` | `tidak magical` |
| 5 | `dab` | `tidak magical` |
| 6 | `abra kadabrac zzzz` | `tidak magical` |

### Note

Cobalah kode ini untuk membantu Anda menjawab soal di atas!

```
char hrf1 = 'b';
char hrf2 = 'g';
int selisih = hrf2 - hrf1
```

## 3. Block Statement

*Block statement* atau dapat juga disebut *compound statement* dapat digunakan untuk mengeksekusi lebih dari satu statement pada suatu cabang dari suatu struktur percabangan. Untuk membuat *block statement,* Anda dapat menempatkan *statement-statement* tersebut di antara pasangan kurung kurawal, '{' dan '}'. Contoh *block statement* dapat dilihat pada Gambar 1.

```
if(x > 0 && x % 2 == 0){
    statement1;
    statement2;
}
```

Gambar 1

Perhatikan Gambar 2! Pada Gambar 2, *statement2* tidak termasuk dalam struktur percabangan *if* sehingga eksekusi *statement* tersebut tidak bergantung kepada kondisi percabangan.

```
if(x > 0 && x % 2 == 0)
    statement1;
    statement2;
```

Gambar 2

## 4. Multiway If-Else Statement

Struktur percabangan *if-else* hanya dapat memiliki dua percabangan, bacalah *slide* dan modul minggu lalu jika Anda lupa. Struktur percabangan yang memiliki lebih dari 2 cabang dapat dituliskan dengan menggunakan struktur *multiway If-else.* Contoh struktur *multiway if-else* dapat dilihat pada Gambar 3.

```
int x = 1;
if(x > 0){
    System.out.println("x merupakan bilangan positif");
}else if(x == 0){
    System.out.println("x bernilai nol");
}else{
    System.out.println("x merupakan bilangan negatif");
}
```

Gambar 3

## 5. Switch-Case Statement

Struktur percabangan *switch-case* merupakan struktur percabangan yang bekerja dengan mengevaluasi nilai dari suatu *controlling expression* dan menyamakan hasilnya dengan beberapa *case label*. Contoh dari struktur ini dapat dilihat pada Gambar 4.

```
int x = 1;
switch (x){
    case 0:
        System.out.println("x bernilai nol");
        break;
    case 1:
        System.out.println("x bernilai satu");
        break;
    default:
        System.out.println("x tidak bernilai nol atau satu");
        break;
}
```

Gambar 4

## 6. Nested-If Statement

Terkadang, penggunaan struktur *multiway if-else* menjadi kurang efisien ketika terdapat beberapa kondisi yang sama pada beberapa branch berbeda, seperti yang ditunjukan Gambar 5.

```
if( x % 2 == 0 && x > 0){
    System.out.println("x adalah bilangan genap positif");
}else if(x % 2 == 0 && x < 0){
    System.out.println("x adalah bilangan genap negatif");
}else if(x % 2 == 1 && x > 0){
    System.out.println("x adalah bilangan ganjil positif");
}else if(x % 2 == 1 && x < 0){
    System.out.println("x adalah bilangan ganjil negatif");
}else{
    System.out.println("x adalah bilangan 0");
}
```

Gambar 5

Kode pada Gambar 5 dapat diubah menjadi struktur *nested-if* seperti yang ditunjukan oleh Gambar 6. Struktur *nested-if*, merupakan suatu struktur di mana percabangan berada di dalam struktur percabangan lain.

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

Gambar 6

## Wombat TV (M0303XXYYY.java)

Televisi di Wombatland terdiri dari 100 channel (0 sampai dengan 99). Sayangnya, walau memiliki banyak *channel*, Wombatland tidak memproduksi *remote*. Mr.Wombat memiliki sebuah televisi produksi Wombatland. Namun, karena Mr.Wombat tidak memiliki remote, maka dia harus memindahkan channel TV secara manual dengan menekan tombol naik dan turun. Dia menyadari bahwa jika channel sudah mencapai 99, dan dia menekan tombol naik, maka channel akan kembali ke nol. Begitupun sebaliknya, jika channel sudah di nol dan dia menekan tombol turun, maka channel akan menjadi 99. Karena Mr.Wombat orang yang malas, dia selalu ingin meminimalkan penekanan tombol. Misalkan, jika *channel* awal adalah 2 dan *channel* tujuan adalah 95, maka dia akan memilih menekan tombol turun sebanyak 7 kali daripada dia menekan tombol naik sebanyak 93 kali. Namun sayang sekali, Mr.Wombat tidak pandai berhitung. Bantulah Mr.Wombat untuk membuatkan program yang menentukan jumlah minimal penekanan tombol serta tombol apa yang harus ditekan.

### Spesifikasi Input

Input berupa dua buah bilangan bulat $a$ dan $b$ ($0 \leq a, b \leq 99$) yang dipisahkan oleh spasi, dimana $a$ adalah *channel* awal dan $b$ adalah *channel* tujuan.

### Spesifikasi Output

Tampilkan *String* dengan format “<jumlah penekanan><spasi><Naik/Turun>”Contoh Input 1

### Contoh Input 1

```
4 15
```

### Contoh Output 1

```
11 Naik
```

### Contoh Input 2

```
99 93
```

### Contoh Output 2

```
6 Turun
```

# JAM

## Deskripsi Masalah

DODO hanya dapat menghitung sampai dengan angka 12. Akibatnya, Dodo tidak dapat membaca waktu yang ditampilkan dengan format 24-jam. Bantulah Dodo dengan cara mengubah jam dari format 24-jam ke format 12-jam. Dalam format 12-jam, satu hari dibagi menjadi dua periode: a.m (ante meridiem/before middday) dan p.m (post meridiem/after middday).

Jadi, hari dimulai pada pukul 12.00 a.m sampai dengan 12.59 a.m., lalu dari 01.00 a.m sampai dengan 11.59 a.m, kemudian diikuti oleh 12.00 p.m. sampai dengan 11.59 p.m. (setelah 12.59 p.m. adalah 01.00 p.m).

## Spesifikasi Masukan

Masukan terdiri dari sebuah bilangan integer (bulat) x (0 x < 24) yang menunjukkan jam dalam format 24-jam.

## Spesifikasi Keluaran

Tampilkan jam dalam format 12-jam. Untuk memudahkan, 12 a.m. ditulis sebagai 0 a.m. dan 12 p.m sebagai 0 p.m.

## Contoh Masukan dan Keluaran

| Masukan | Keluaran yang Diharapkan |
|---|---|
| 5 | 5 a.m. |
| 14 | 2 p.m. |