# Bab 5. Struktur Perulangan

**OBJEKTIF :**

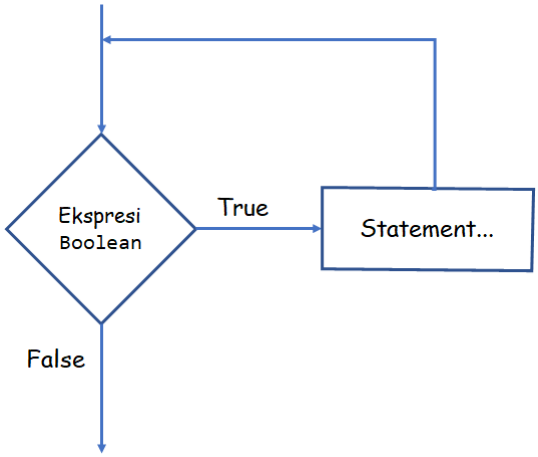
1. Mahasiswa mampu memahami mengenai Struktur Perulangan pada Java.
2. Mahasiswa mampu memahami mengenai penggunaan Struktur Perulangan pada program Java.
3. Mahasiswa mampu mensimulasikan penggunaan Struktur Perulangan pada program Java untuk kejadian di dunia nyata.

---

## 5.1 *Loop* `while`

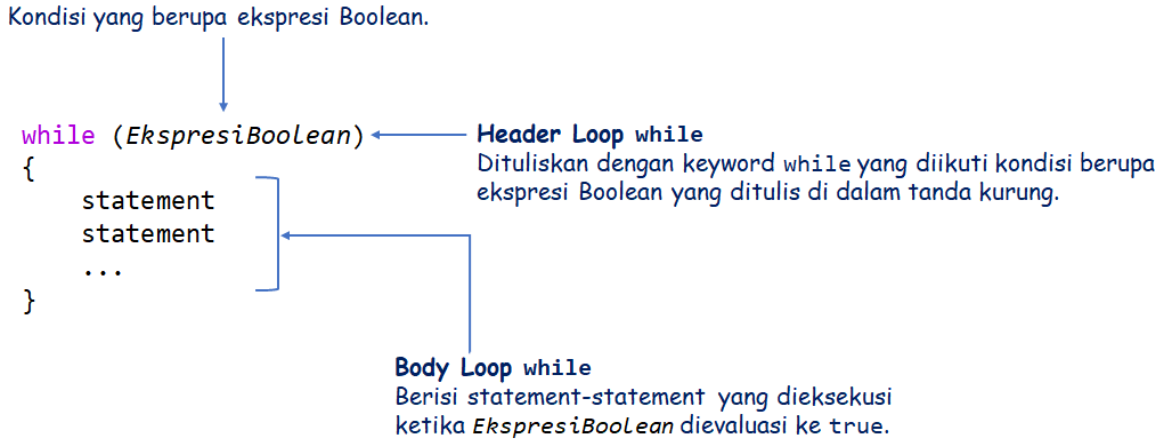
Kita dapat membuat program melakukan eksekusi sekelompok *statement* secara berulang dengan menuliskan struktur perulangan. Struktur perulangan ini sering disebut dengan *loop*. Java mempunyai tiga *statement loop* yaitu: *loop* `while`, *loop* `do-while`, dan *loop* `for`. Pada bagian ini kita akan membahas *loop* `while`.

*Loop* `while` digunakan untuk membuat program mengulang eksekusi dari sekelompok *statement-statement* selama suatu kondisi terpenuhi. Logika dari *loop* `while` ditunjukkan oleh gambar berikut.




Dalam *loop* `while`, sekelompok *statement-statement* dieksekusi secara berulang kali selama kondisi yang berupa ekspresi *Boolean* bernilai `true`. Pengulangan eksekusi sekelompok *statement-statement* ini berhenti ketika ekspresi `Boolean` bernilai `false`.

*Syntax* penulisan *loop* `while` adalah sebagai berikut.



Baris pertama dari *loop* `while` adalah header dari *loop*. *Header* dari *loop* `while` dituliskan dengan *keyword* `while` yang diikuti kondisi berupa ekspresi *Boolean* yang ditulis di dalam tanda kurung. Header dari *loop* diikuti dengan body dari *loop* yang dituliskan di dalam tanda kurung kurawal. *Body* dari *loop* ini berisikan satu atau lebih *statement-statement* yang dieksekusi jika ekspresi *Boolean* pada *header* dievaluasi ke `true`.

*Loop* `while` bekerja seperti *statement* `if` yang dieksekusi berkali-kali. Selama ekspresi *Boolean* yang berada di dalam tanda kurung bernilai `true`, *statement-statement* di dalam *body loop* `while` dieksekusi. Program berikut mencontohkan penggunaan *loop* `while` untuk mencetak teks `Halo` sebanyak lima kali:

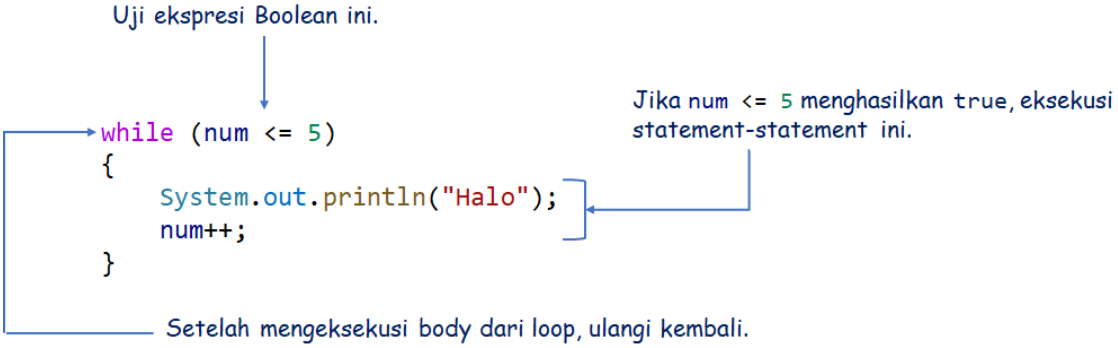
***Program (LoopWhile.java)***

```
/*
    Program ini menggunakan loop while untuk
    mencetak Halo sebanyak lima kali.
*/
public class LoopWhile
{
    public static void main(String[] args)
    {
        int num = 1;

        while (num <= 5)
        {
            System.out.println("Halo");
            num++;
        }

        System.out.println("Selesai!");
    }
}
```

***Output Program (LoopWhile.java)***

```
Halo
Halo
Halo
Halo
Halo
Selesai!
```

Gambar berikut menjelaskan *statement* `while` pada program di atas.

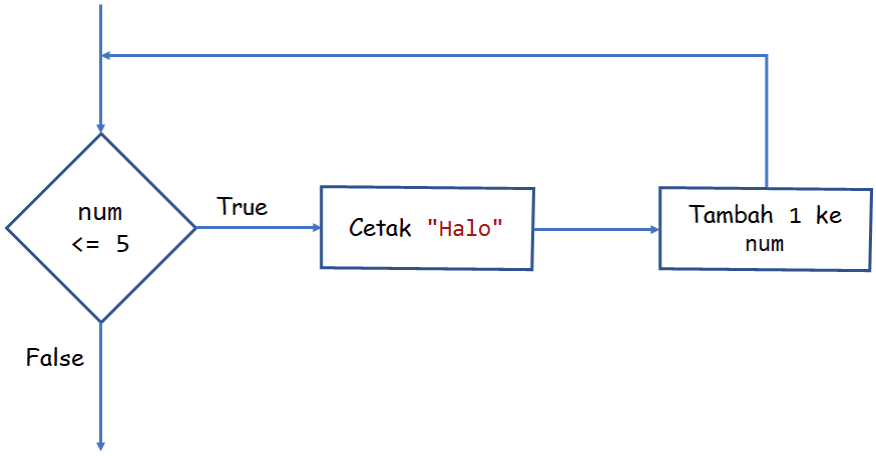


Setiap pengulangan dari *loop* disebut sebagai *iterasi*. *Loop* pada program di atas, melakukan lima iterasi karena variabel `num` diinisialisasi dengan nilai 1, dan nilainya diinkrementasi setiap kali *body* dari *loop* dieksekusi. Pada iterasi ke-5, nilai `num` menjadi 6 yang menyebabkan ekspresi *Boolean* `num <= 5` dievaluasi ke `false`, sehingga menyebabkan *loop* berhenti dan program melanjutkan *ekseksi* ke *statement* setelah *loop*.

Tabel berikut menunjukkan perubahan nilai variabel `num`, hasil uji ekspresi *Boolean*, dan aksi yang dilakukan pada setiap iterasi dari *loop* `while` di atas:

| Iterasi | num | num <= 5 | Eksekusi |
|---|---|---|---|
| 1 | 1 | True | - Cetak "Halo"<br>- num++ menghasilkan num = 2 |
| 2 | 2 | True | - Cetak "Halo"<br>- num++ menghasilkan num = 3 |
| 3 | 3 | True | - Cetak "Halo"<br>- num++ menghasilkan num = 4 |
| 4 | 4 | True | - Cetak "Halo"<br>- num++ menghasilkan num = 5 |
| 5 | 5 | True | - Cetak "Halo"<br>- num++ menghasilkan num = 6 |
| 6 | 6 | False | **BERHENTI** |

*Flowchart* berikut mengilustrasikan logika dari *loop* `while` pada contoh di atas:




## Contoh Program Menggunakan *Loop* `while`

Misalkan kita membuat program yang menghitung berapa lama suatu deposito bank harus disimpan sehingga saldo akhir menjadi dua kali lipat dari saldo awal. Misalkan bunga deposito sebesar 10% setiap tahunnya dan diakumulasikan ke saldo deposito. Program menanyakan pengguna jumlah saldo awal deposito lalu menghitung berapa lama deposito harus disimpan untuk saldo akhirnya menjadi dua kali lipat dari saldo awal.

Kita dapat menuliskan *pseudocode* untuk program seperti berikut:

```
Minta pengguna memasukkan nilai saldo_awal
Mulai dengan tahun 0 dan tetapkan saldo_berjalan sama dengan nilai saldo_awal
Ulangi langkah-langkah berikut selama saldo_berjalan kurang dari 2 * saldo awal:
    Tambah 1 ke tahun
    Hitung jumlah bunga dengan saldo_berjalan x 0.1
    Tambahkan jumlah bunga ke saldo_berjalan
Tetapkan tahun terakhir sebagai tahun dimana saldo akhir sama dengan saldo_awal.
```

Jika kita melakukan perhitungan manual dari *pseudocode* di atas, kita dapat melakukannya menggunakan tabel seperti berikut (misalkan pengguna memasukkan saldo awal 1.000.000):

| Tahun | Bunga | Saldo Berjalan |
|---|---|---|
| 0 | 0 | 1,000,000.00 |
| 1 | 1,000,000.00 x 0.1 = 100,000.00 | 1,100,000.00 |
| 2 | 1,100,000.00 x 0.1 = 110,000.00 | 1,210,000.00 |
| 3 | 1,210,000.00 x 0.1 = 121,000.00 | 1,331,000.00 |
| 4 | 1,331,000.00 x 0.1 = 133,100.00 | 1,464,100.00 |
| 5 | 1,464,100.00 x 0.1 = 146,410.00 | 1,610,510.00 |
| 6 | 1,610,510.00 x 0.1 = 161,051.00 | 1,771,561.00 |
| 7 | 1,771,561.00 x 0.1 = 177,156.10 | 1,948,717.10 |
| 8 | 1,948,717.10 x 0.1 = 194,871.71 | 2,143,588.81 ← Berhenti! Saldo Berjalan >= 2 * Saldo Awal. |

Program yang mengimplementasikan algoritma di atas dapat dituliskan seperti program berikut:

***Program (Deposito.java)***

```java
import java.util.Scanner;

/*
    Program ini menghitung berapa tahun setoran awal
    dari sebuah deposito menjadi dua kali lipat.
*/
public class Deposito {
    public static void main(String[] args)
    {
        final double PERSEN_BUNGA = 0.10;   // Bunga = 10%
        double saldoAwal;                   // Untuk menyimpan saldo awal
        double saldoBerjalan;               // Untuk menyimpan saldo berjalan
        double bunga;                       // Untuk mneyimpan jumlah bunga
        int tahun;                          // Untuk menyimpan tahun

        Scanner keyboard = new Scanner(System.in);

        // Minta pengguna memasukkan nilai setoran
        System.out.print("Masukkan saldo awal deposito: ");
        saldoAwal = keyboard.nextDouble();

        // Tetapkan tahun mulai dengan 0 dan saldo dengan setoran
        tahun = 0;
        saldoBerjalan = saldoAwal;

        // Loop untuk menghitung saldo berjalan
        while (saldoBerjalan <= 2 * saldoAwal)
        {
            tahun++;
```

```
            bunga = PERSEN_BUNGA * saldoBerjalan;
            saldoBerjalan = saldoBerjalan + bunga;
        }

        System.out.print("Deposito Anda akan menjadi dua kali lipat");
        System.out.println(" setelah: " + tahun + " tahun.");
        System.out.printf("Saldo akhir = Rp.%,.2f\n", saldoBerjalan);
    }
}
```

***Output Program (Deposito.java)***

```
Masukkan saldo awal deposito: 1000000
Deposito Anda akan menjadi dua kali lipat setelah: 8 tahun.
Saldo akhir = Rp.2,143,588.81
```

## *Infinite loop*

Dalam semua kecuali kasus yang sangat jarang, *loop* harus mempunyai sebuah cara untuk menghentikan iterasi. Ini berarti di dalam *body loop* `while` harus terdapat sebuah *statement* yang dapat mengubah hasil evaluasi dari ekspresi *Boolean* yang diuji ke `false` untuk menghentikan *loop*. *Loop* pada contoh program `LoopWhile.java` mempunyai *statement* inkrementasi, `num++`, yang menyebabkan *loop* berhenti setelah lima iterasi.

Tanpa *statement* yang dapat mengubah hasil evaluasi kondisi, *loop* `while` akan mengulang terus menerus dan tidak akan pernah berhenti. *Loop* yang tidak berhenti ini disebut dengan *infinite loop* (*infinite* berarti tak hingga).

Perhatikan kode berikut:

```
int num = 1;
while (num <= 5)
{
    System.out.println("Halo");
}
```

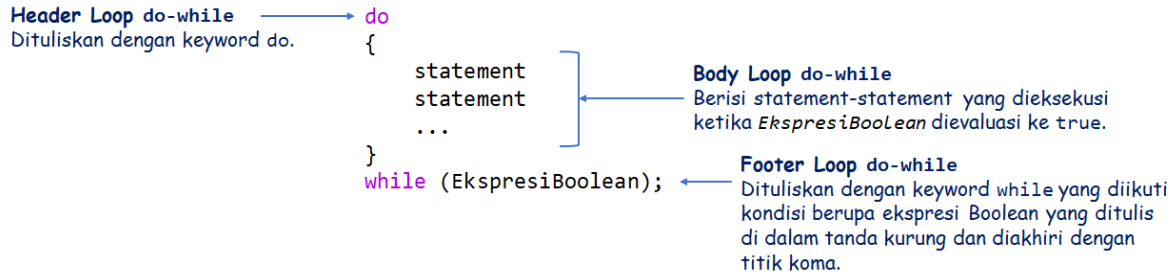
*Loop* di atas adalah *infinite loop* karena tidak terdapat *statement* yang mengubah variabel `num` yang diuji dalam kondisi *loop* tersebut. Setiap kali ekspresi *Boolean*, `num <= 5` diuji, `num` akan selalu bernilai 1 dan ekspresi *Boolean* tersebut akan selalu dievaluasi ke `true`.

Dalam menulis program kita umumnya menghindari *infinite loop*. Hanya dalam kasus tertentu yang jarang sekali kita memerlukan *infinite loop*. Jika program kita tidak sengaja berjalan dengan *infinite loop* karena kesalahan kode, kita dapat menghentikan jalannya program dengan menginterupsi program dengan menekan CTRL + C.

# 5.2 *Loop* `do-while`

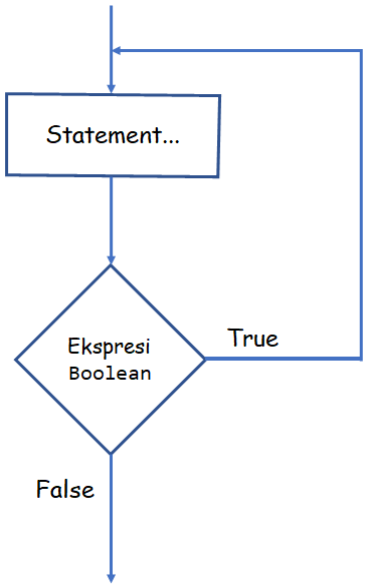
*Loop* `do-while` adalah *loop* `while` terbalik. *Loop* `do-while` menguji kondisi setelah mengeksekusi *statement-statement* dalam *body loop*, mengulang eksekusi jika kondisi bernilai `true`, dan keluar dari *loop* jika kondisi bernilai `false`.

Berikut adalah *syntax* dari *loop* `do-while`:




Perhatikan bahwa penulisan *footer loop* `do-while` diakhiri dengan titik koma.

Logika dari *loop* `do-while` digambarkan dalam *flowchart* berikut:




Perbedaan *loop* `do-while` dengan *loop* `while` adalah pada *loop* `do-while`, *statement-statement* dalam *body loop* dieksekusi terlebih dahulu sebelum kondisi diuji. *Loop* `while` sering disebut sebagai *loop pre-test* karena pengujian kondisi dilakukan sebelum mengeksekusi *body* dari *loop*, sedangkan *loop* `do-while` sering disebut sebagai *loop post-test* karena pengujian kondisi dilakukan setelah mengeksekusi *body* dari *loop*.

Pada *loop* `do-while` *body* dari *loop* dieksekusi setidaknya sekali, meskipun ekspresi *Boolean* bernilai `false` saat *loop* dimulai. Sebagai contoh, dalam *loop* `while` berikut *statement* `println` tidak akan dieksekusi sama sekali:

```
int x = 1;
while (x < 0)
{
    System.out.println(x);
}
```

Sebaliknya, *statement* `println` dalam *loop* `do-while` berikut akan dieksekusi sekali karena *loop* `do-while` tidak mengevaluasi ekspresi `x < 0` sampai akhir dari iterasi:

```
int x = 1;
do
{
    System.out.println("Halo");
}
while (x < 0);
```

Gambar berikut mengilustrasikan perbedaan cara kerja *loop* `while` dan *loop* `do-while` dari contoh di atas.

```
int x = 1;
while (x < 0)
{
    System.out.println("Halo");
    x++;
}
```

***Output***

```
```

Program tidak mencetak `Halo` sama sekali karena ketika loop dimulai, kondisi `x < 0` menghasilkan `false`, sehingga statement-statement dalam body loop tidak dieksekusi sama sekali.

```
int x = 1;
do
{
    System.out.println("Halo");
    x++;
}
while (x < 0);
```

***Output***

```
Halo
```

Program mencetak `Halo` satu kali, karena ketika loop dimulai, statement-statement dalam body loop dieksekusi terlebih dahulu sebelum kondisi `x < 0` diuji. Dan karena `x < 0` menghasilkan false saat iterasi pertama, maka loop berhenti.

Kita menggunakan *loop* `do-while` ketika kita menginginkan *loop* setidaknya dieksekusi sekali. Sebagai contoh, program berikut menghitung rata-rata dari tiga skor ujian dari seorang siswa. Setelah rata-rata ditampilkan, program menanyakan pengguna apakah ingin menghitung rata-rata dari tiga ujian yang lain. Program berulang selama pengguna mengetikkan `Y`.

***Program (RerataUjianLoop.java)***

```
import java.util.Scanner;

/*
    Program ini meminta pengguna memasukkan tiga nilai ujian lalu
    menghitung rata-rata tiga nilai ujian yang dimasukkan.
    Kemudian program menanyakan pengguna apakah ingin memasukkan
    tiga nilai ujian lainnya. Jika ya, program mengulang dan jika tidak,
    program berhenti.
*/
public class RerataUjianLoop
{
    public static void main(String[] args)
    {
        double skor1, skor2, skor3; // Untuk menyimpan tiga skor ujian
        double rerata;              // Untuk menyimpan rata-rata
        String input;               // Untuk menyimpan input
        char ulangi;                // Untuk menyimpan 'y' atau 'n'

        Scanner keyboard = new Scanner(System.in);

        do
        {
```

```
            // Ambil skor ujian 1
            System.out.print("Masukkan skor ujian 1: ");
            skor1 = keyboard.nextDouble();

            // Ambil skor ujian 2
            System.out.print("Masukkan skor ujian 2: ");
            skor2 = keyboard.nextDouble();

            // Ambil skor ujian 3
            System.out.print("Masukkan skor ujian 3: ");
            skor3 = keyboard.nextDouble();

            // Lewati karakter baris baru
            keyboard.nextLine();

            // Hitung rata-rata dan tampilkan hasilnya
            rerata = (skor1 + skor2 + skor3) / 3;
            System.out.printf("Rata-rata ujian = %.2f\n", rerata);

            // Tanya pengguna apakah ingin melanjutkan menghitung rata-rata
            // dari tiga skor ujian lainnya.
            System.out.println("Apakah Anda ingin menghitung rata-rata" +
                               "tiga skor ujian lainnya?");
              System.out.print("Ketik Y untuk melanjutkan dan N untuk berhenti:
");
            input = keyboard.nextLine();
            ulangi = input.charAt(0);
        }
        while (ulangi == 'Y' || ulangi == 'y');
    }
}
```

***Output Program (RerataUjianLoop.java)***

```
Masukkan skor ujian 1: 56.7
Masukkan skor ujian 2: 78.6
Masukkan skor ujian 3: 98.66
Rata-rata ujian = 77.99
Apakah Anda ingin menghitung rata-ratatiga skor ujian lainnya?
Ketik Y untuk melanjutkan dan N untuk berhenti: y
Masukkan skor ujian 1: 56.77
Masukkan skor ujian 2: 54.78
Masukkan skor ujian 3: 98.66
Rata-rata ujian = 70.07
Apakah Anda ingin menghitung rata-ratatiga skor ujian lainnya?
Ketik Y untuk melanjutkan dan N untuk berhenti: N
```

## 5.3 *Loop* `for`

*Loop* `while` dan *loop* `do-while` adalah *loop* yang berdasarkan suatu kondisi. *Loop* `for` adalah *loop* yang pengulangannya sejumlah tertentu.

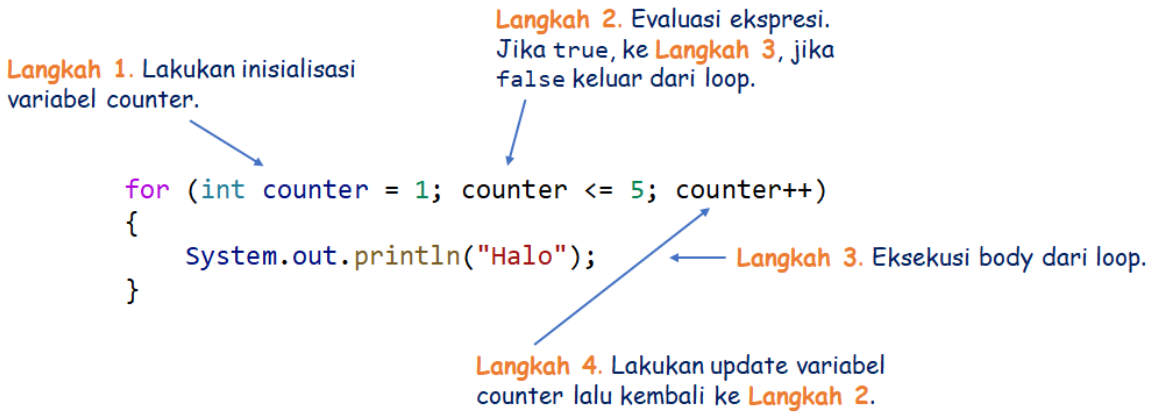
*Loop* `for` harus mempunyai tiga elemen berikut:

1. **Inisialisasi** variabel *counter* (pencacah) yang menghitung banyaknya *loop*.
2. **Penguji Batas** yang menguji variabel *counter* terhadap suatu batas (maksimum ataupun minimum). Ketika variabel *counter* mencapai batas, *loop* berhenti.
3. **Pengupdate** yang menambahkan atau mengurangi nilai variabel *counter* setiap iterasi. Umumnya variabel *counter* diinkrementasi.

Berikut adalah contoh *loop* `for` yang mencetak `Halo` sebanyak lima kali:

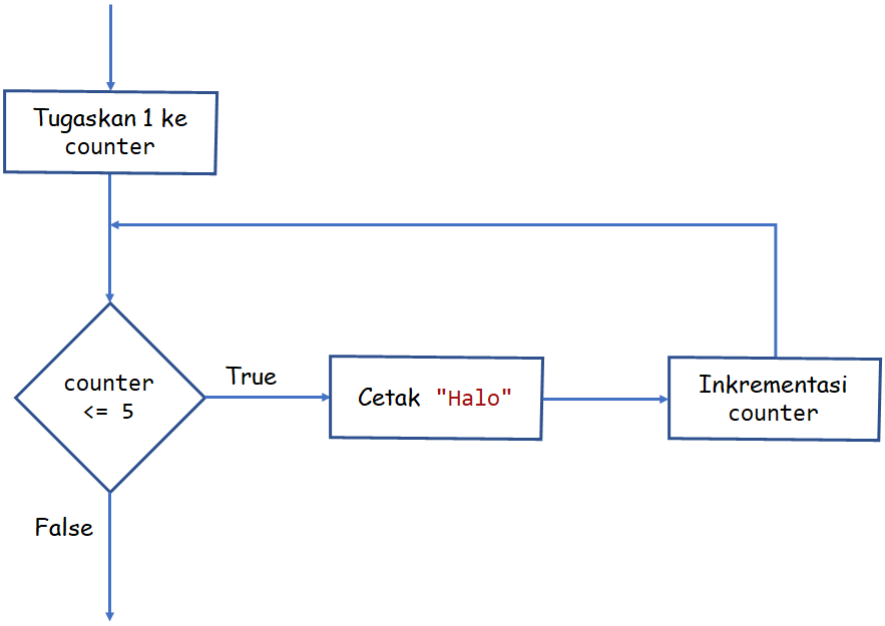
```
for (int counter = 1; counter <= 5; counter++)
{
    System.out.println("Halo");
}
```

Pada contoh *loop* `for` di atas, `counter = 1` adalah *inisiliasi*, `counter <= 5` adalah kondisi, dan `counter++` adalah *pengupdate* yang berupa *statement inkrementasi* (menambahkan 1 ke `counter`). *Body* dari *loop* `for` di atas hanya mempunyai satu *statement* yaitu `System.out.println("Halo");`. Gambar berikut mengilustrasikan langkah-langkah pemrosesan saat *loop* `for` di atas dieksekusi:

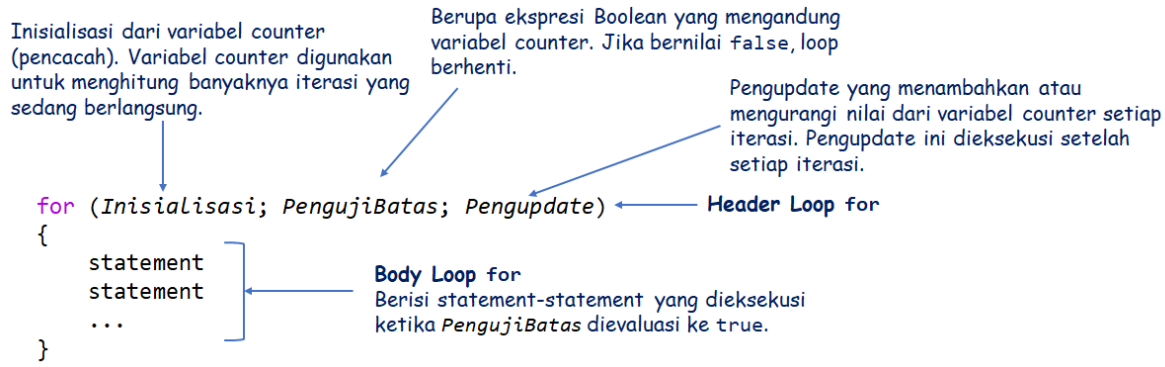



Perhatikan Langkah 2 sampai dengan Langkah 4 diulang selama kondisi bernilai `true`.

Logika dari *loop* `for` dari contoh di atas ditunjukkan pada *flowchart* berikut.




*Syntax* penulisan *loop* `for` adalah sebagai berikut:




Variabel *counter* atau pencacah yang diinisialisasi pada *header loop* dapat digunakan oleh *statement-statement* dalam *body loop*. Sebagai contoh, pada kode berikut terdapat *statement* di dalam *body loop* yang mencetak nilai variabel *counter* bernama `num`:

```
for (int num = 1; num <= 10; num++)
{
    System.out.println("Iterasi ke-" + num);
}
```

*Output* dari kode di atas:

```
Iterasi ke-1
Iterasi ke-2
Iterasi ke-3
Iterasi ke-4
Iterasi ke-5
Iterasi ke-6
Iterasi ke-7
Iterasi ke-8
Iterasi ke-9
Iterasi ke-10
```

Variabel *counter* yang kita *inisialisasi* di dalam *header loop* `for` hanya dapat digunakan oleh *statement-statement* dalam dalam *body loop* dan tidak dapat digunakan pada *statement* di luar *body loop*. Kode berikut akan menghasilkan *error* karena mencoba menggunakan variabel *counter* di luar *loop*:

```
for (int num = 1; num <= 10; num++)
{
    System.out.println("Iterasi ke-" + num);
}

System.out.println(num);     // ERROR! num hanya tersedia di dalam loop.
```

Kita juga dapat menginisialisasi variabel *counter* di luar *header* dari *loop* `for`, seperti pada contoh berikut:

```
int num;

for (num = 1; num <= 10; num++)
{
    System.out.println("Iterasi ke-" + num);
}
```

Perhatikan pada kode di atas, kita mendeklarasikan variabel `num` yang digunakan sebagai variabel *counter* sebelum *loop*. Lalu, pada inisialisasi dalam *header loop*, kita tidak menuliskan deklarasi inisialisasi namun menuliskan penugasan variabel `num` dengan suatu nilai.

Jika kita mendeklarasikan variabel *counter* di luar *loop*, kita dapat menggunakannya setelah *loop*, seperti dicontohkan pada kode berikut:

```
int num;

for (num = 1; num <= 10; num++)
{
    System.out.println("Iterasi ke-" + num);
}

System.out.println("Nilai num = " + num);
```

Kode di atas akan menghasilkan *output* seperti berikut:

```
Iterasi ke-1
Iterasi ke-2
Iterasi ke-3
Iterasi ke-4
Iterasi ke-5
Iterasi ke-6
Iterasi ke-7
Iterasi ke-8
Iterasi ke-9
Iterasi ke-10
Nilai num = 11
```

## Bentuk Lain Pengupdate

Pengupdate dalam *loop* `for` tidak harus menginkrementasi variabel *counter*. Berikut adalah contoh dari *loop* yang menampilkan angka genap dari 2 sampai dengan 100 dengan menambahkan 2 ke variabel *counter*:

```
for (num = 2; num <= 100; num += 2)
{
    System.out.println(num);
}
```

Kode berikut adalah *loop* yang menghitung mundur dari 10 ke 0:

```
for (num = 10; num >= 0; num--)
{
    System.out.prinln(num);
}
```

## Contoh Program yang Menggunakan *Loop* `for`

Misalkan kita membuat sebuah program yang mencetak sebuah tabel dari nilai suatu deposito per tahun dalam suatu jangka waktu. Program meminta pengguna untuk memasukkan setoran deposito dan jangka waktu penyimpanan yang diinginkan. Program akan mencetak saldo deposito per tahunnya dalam jangka waktu yang dimasukkan pengguna.

*Pseudocode* dari program dapat dituliskan seperti berikut:

```
Input setoran deposito
Input jangka waktu deposito
Ulangi langkah berikut selama tahun lebih kecil atau sama dengan jangka waktu:
    Hitung bunga dengan rumus saldo * PERSEN_BUNGA / 100
    Hitung saldo sekarang dengan menambahkan saldo sebelumnya dengan bunga
    Tampilkan tahun dan saldo setelah bunga
    Inkrementasi tahun
```

***Program (TabelDeposito.java)***

```
import java.util.Scanner;

/*
    Program ini mencetak tabel saldo deposito dengan bunga 5% dari
    setoran deposito dan jangka waktu yang diinput pengguna.
*/
public class TabelDeposito
{
    public static void main(String[] args)
    {
        final double PERSEN_BUNGA = 5;      // Bunga deposito 5% per tahun
         int jangkaWaktu;                        // Untuk menyimpan jangka waktu
deposito
        double saldo;                       // Untuk menyimpan saldo deposito

        Scanner keyboard = new Scanner(System.in);
```

```
        // Minta nilai setoran deposito
        System.out.print("Masukkan setoran deposito: ");
        saldo = keyboard.nextDouble();

        // Minta jangka waktu deposito
        System.out.print("Jangka waktu deposito (tahun): ");
        jangkaWaktu = keyboard.nextInt();

        // Cetak tabel nilai deposito per tahun selama jangka waktu
        // yang dimasukkan penggunna
        for (int tahun = 1; tahun <= jangkaWaktu; tahun++)
        {
            double bunga = saldo * PERSEN_BUNGA / 100;
            saldo = saldo + bunga;
            System.out.printf("%4d %,20.2f\n", tahun, saldo);
        }
    }
}
```

***Output Program (TabelDeposito.java)***

```
Masukkan setoran deposito: 1000000
Jangka waktu deposito (tahun): 10
   1         1,050,000.00
   2         1,102,500.00
   3         1,157,625.00
   4         1,215,506.25
   5         1,276,281.56
   6         1,340,095.64
   7         1,407,100.42
   8         1,477,455.44
   9         1,551,328.22
  10         1,628,894.63
```

## 5.4 Pengaplikasian *Loop* pada Program

Pada bagian ini kita akan membahas penggunaan *loop* yang umum pada program-program. Kita akan membahas penggunaan *loop* dalam program untuk melakukan hal-hal berikut:

- Validasi *Input*. *Loop* dapat digunakan untuk memastikan pengguna memasukkan *input* yang diminta. *Loop* akan meminta pengguna memasukkan ulang *input* jika *input* yang dimasukkan salah.
- Mendapatkan Total dan Rata-rata. *Loop* dapat digunakan untuk program yang meminta pengguna memasukkan serangkaian angka dan menghitung total jumlah dan rata-rata dari angka-angka yang dimasukkan.
- Nilai *Sentinel*. *Loop* dapat digunakan untuk meminta pengguna memasukkan serangkaian data sampai dengan pengguna memasukkan sebuah nilai tertentu yang digunakan untuk menghentikan *input*. Nilai tertentu yang menghentikan *input* ini disebut nilai *sentinel* (*sentinel* berarti pensinyal).
- Mencari Maksimum dan Minimum. *Loop* dapat digunakan untuk mencari nilai maksimum dan minimum dari serangkaian nilai-nilai.

## Validasi *Input*

Validasi *input* adalah proses pemeriksaan data yang diberikan ke program oleh pengguna dan menentukan apakah data yang diberikan adalah valid. Program yang baik haruslah memberikan petunjuk jelas mengenai jenis *input* yang diterima, dan tidak mengasumsikan pengguna mengikuti petunjuk-petunjuk tersebut.

*Loop* `while` dapat digunakan untuk melakukan validasi *input*. Jika nilai invalid (tidak benar) dimasukkan, *loop* dapat meminta pengguna untuk memasukkan nilai lain sebanyak mungkin sampai nilai yang benar dimasukkan. Sebagai contoh, misalkan sebuah *loop* yang meminta pengguna untuk memasukkan angka di antara 1 sampai dengan 100:

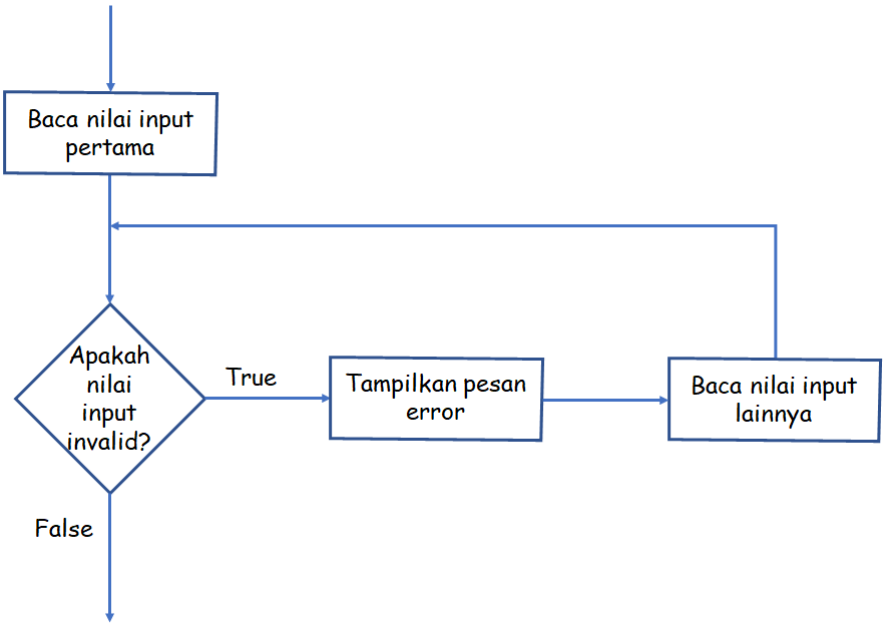
***Program (ValidasiInput.java)***

```
import java.util.Scanner;

/*
    Program ini mendemonstrasikan validasi input
    menggunakan loop while
*/
public class ValidasiInput
{
    public static void main(String[] args)
    {
        int num;      // Untuk menyimpan angka input

        Scanner keyboard = new Scanner(System.in);

        // Minta angka bulat 1 s.d 100
        System.out.print("Masukkkan angka bulat antara 1 s.d 100: ");
        num = keyboard.nextInt();

        // Minta pengguna memasukkan angka bulat 1 s.d 100 kembali
        // jika pengguna memasukkan angka yang salah.
        while (num < 1 || num > 100)
        {
            System.out.print("Angka yang Anda masukkan salah. ");
            System.out.println("Silahkan masukkan kembali.");
            System.out.print("Masukkkan angka bulat antara 1 s.d 100: ");
            num = keyboard.nextInt();
        }

        System.out.println("Angka yang Anda masukkan adalah " + num);
    }
}
```

***Output Program (ValidasiInput.java)***

```
Masukkkan angka bulat antara 1 s.d 100: 0
Angka yang Anda masukkan salah. Silahkan masukkan kembali.
Masukkkan angka bulat antara 1 s.d 100: 102
Angka yang Anda masukkan salah. Silahkan masukkan kembali.
Masukkkan angka bulat antara 1 s.d 100: 80
Angka yang Anda masukkan adalah 80
```

Logika dari validasi *input* dapat dilihat pada *flowchart* berikut.




## Mendapatkan Total dan Rata-rata

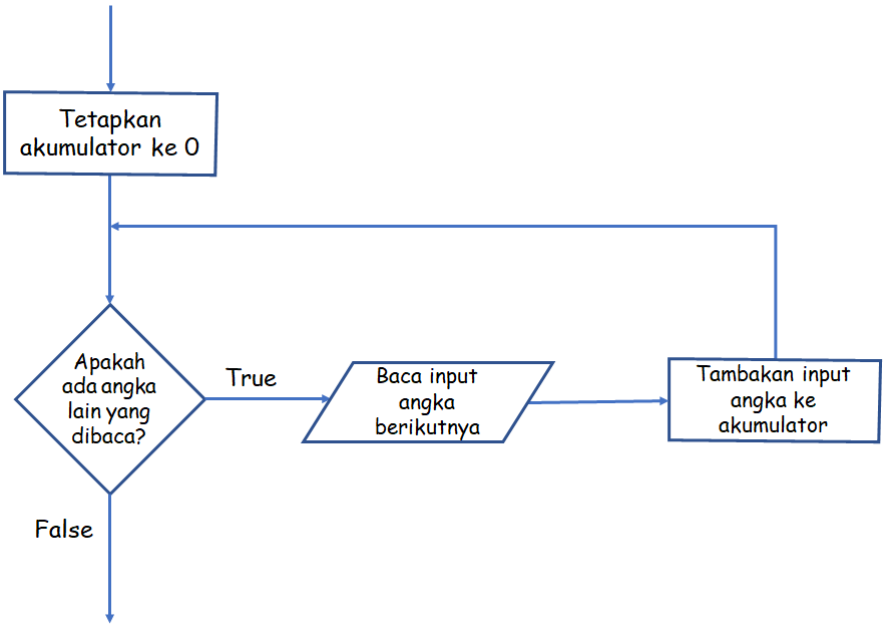
*Loop* dapat digunakan untuk menghitung total dari *input-input* yang diberikan pengguna. Misalkan, kita menuliskan sebuah program yang menghitung total jumlah penjualan dalam satu minggu. Program meminta *input* pengguna penjualan per hari dan menghitung total dari penjualan-penjualan tersebut.

Program yang menghitung total dari rangkaian angka-angka umumnya menggunakan dua elemen berikut:

- Sebuah *loop* yang membaca setiap angka dalam rangkaian
- Sebuah variabel yang mengakumulasikan total jumlah dari angka-angka saat dibaca.

Variabel yang digunakan untuk mengakumulasikan total jumlah dari angka-angka disebut dengan akumulator. *Loop* yang mengakumulasikan total jumlah ini sering disebut sebagai *loop* yang menghitung total berjalan karena *loop* tersebut mengakumulasikan total saat membaca setiap angka dalam rangkaian.

*Flowchart* dari *loop* yang menghitung total berjalan ditunjukkan pada gambar berikut:



Ketika *loop* selesai, variabel akumulator akan menyimpan total jumlah angka-angka yang dibaca oleh *loop*. Perhatikan pada langkah pertama dalam *flowchart* adalah penetapan variabel akumulator dengan 0. Langkah ini sangat penting. Setiap kali *loop* membaca sebuah angka, *loop* tersebut menambahkan angka yang dibaca ke akumulator. Jika akumulator dimulai dengan angka selain 0, maka akumulator tersebut tidak akan berisi total jumlah yang benar ketika *loop* selesai.

Mari kita lihat contoh program yang menghitung total berjalan. Program berikut menghitung total jumlah penjualan selama periode yang ditentukan pengguna dan meminta pengguna memasukkan penjualan per harinya lalu menghitung total berjalan saat penjualan per hari diterima. Setelah mendapatkan semua angka penjualan, program mencetak total penjualan dan rata-rata penjualan per hari.

***Program (TotalPenjualan.java)***

```
import java.util.Scanner;

/*
    Program ini menggunakan loop untuk menghitung total berjalan
    dan rata-rata dari nilai-nilai yang dimasukkan pengguna.
*/
public class TotalPenjualan
{
    public static void main(String[] args)
    {
        int banyakHari;             // Banyak hari penjualan
        double penjualan;           // Penjualan per hari
        double totalPenjualan;      // Akumulator
        double rataRataPenjualan;   // Rata-rata penjualan

        Scanner keyboard = new Scanner(System.in);

        // Tanya pengguna banyaknya hari input penjualan
        System.out.print("Berapa hari Anda ingin menginput penjualan? ");
        banyakHari = keyboard.nextInt();

        // Inisialisasi total penjualan dengan 0
        totalPenjualan = 0.0;

        // Loop untuk menanyakan penjualan per hari dan menjumlahkan
        // penjualan ke variabel akumulator totalPenjualan.
        for (int hari = 1; hari <= banyakHari; hari++)
        {
            System.out.print("Masukkan penjualan hari ke-" + hari + " : ");
            penjualan = keyboard.nextDouble();
            totalPenjualan = totalPenjualan + penjualan;
        }

        // Tampilkan total penjualan
        System.out.printf("Total penjualan = Rp.%,.2f\n", totalPenjualan);

        // Tampilkan rata-rata penjualan per hari
        rataRataPenjualan = totalPenjualan / banyakHari;
              System.out.printf("Rata-rata  penjualan  per  hari  =  Rp.%,.2f\n",
rataRataPenjualan);
    }
}
```

***Output Program (TotalPenjualan.java)***

```
Berapa hari Anda ingin menginput penjualan? 7
Masukkan penjualan hari ke-1 : 1250000
Masukkan penjualan hari ke-2 : 2500000
Masukkan penjualan hari ke-3 : 3500000
Masukkan penjualan hari ke-4 : 5450000
Masukkan penjualan hari ke-5 : 4750000
Masukkan penjualan hari ke-6 : 6750000
Masukkan penjualan hari ke-7 : 8500000
Total penjualan = Rp.32,700,000.00
Rata-rata penjualan per hari = Rp.4,671,428.57
```

## Nilai *Sentinel*

Pada contoh program yang menghitung total berjalan, di awal program, program menanyakan pengguna berapa *input* yang ingin dimasukkan. Terkadang pengguna mempunyai daftar nilai *input* yang sangat panjang dan tidak mengetahui berapa banyak angka yang ingin dimasukkan. Teknik untuk menulis program dengan kasus seperti ini adalah dengan meminta pengguna untuk memasukkan sebuah nilai tertentu untuk mengindikasikan bahwa pengguna ingin mengakhiri *input*. Nilai tertentu ini disebut dengan nilai *sentinel*.

Misalkan kita memodifikasi program `TotalPenjualan.java` pada bagian sebelumnya dengan menerapkan nilai *sentinel*. Kode program ini dapat dituliskan seperti berikut:

***Program (TotalPenjualanSentinel.java)***

```java
import java.util.Scanner;

/*
    Program ini mendemonstrasikan penggunaan
    nilai sentinel untuk mengakhiri input pengguna.
*/
public class TotalPenjualanSentinel
{
    public static void main(String[] args)
    {
        double penjualan;           // Penjualan per hari
        double totalPenjualan;      // Akumulator

        Scanner keyboard = new Scanner(System.in);

        // Inisialisasi total penjualan dengan 0
        totalPenjualan = 0.0;

        // Minta input penjualan pertama
        System.out.print("Masukkan penjualan (-1 untuk mengakhiri): ");
        penjualan = keyboard.nextDouble();

        // Loop untuk menanyakan penjualan per hari dan menjumlahkan
        // penjualan ke variabel akumulator totalPenjualan.
        while (penjualan != -1 )
        {
            totalPenjualan = totalPenjualan + penjualan;

            // Minta input penjualan
```

```
            System.out.print("Masukkan penjualan (-1 untuk mengakhiri): ");
            penjualan = keyboard.nextDouble();
        }

        // Tampilkan total penjualan
        System.out.printf("Total penjualan = Rp.%,.2f\n", totalPenjualan);
    }
}
```

***Output Program (TotalPenjualanSentinel.java)***

```
Masukkan penjualan (-1 untuk mengakhiri): 1250000
Masukkan penjualan (-1 untuk mengakhiri): 2500000
Masukkan penjualan (-1 untuk mengakhiri): 3500000
Masukkan penjualan (-1 untuk mengakhiri): 5450000
Masukkan penjualan (-1 untuk mengakhiri): 4750000
Masukkan penjualan (-1 untuk mengakhiri): 6750000
Masukkan penjualan (-1 untuk mengakhiri): 8500000
Masukkan penjualan (-1 untuk mengakhiri): -1
Total penjualan = Rp.32,700,000.00
```

## Mencari Maksimum dan Minimum

Algoritma *loop* yang umum juga adalah untuk mencari nilai maksimum dan minimum dari serangkaian angka-angka.

***Program (MaxMin.java)***

```
import java.util.Scanner;

/*
    Program ini mencari nilai maksimum dan minimum dari
    rangkaian angka-angka yang dimasukkan pengguna.
*/
public class MaxMin
{
    public static void main(String[] args)
    {
        int max, min;
        int input;

        Scanner keyboard = new Scanner(System.in);

        System.out.print("Masukkan angka (-1 untuk mengakhiri): ");
        input = keyboard.nextInt();

        max = input;
        min = input;

        while (input != -1)
        {
            if (max < input)
            {
                max = input;
            }
```

```
            if (min > input)
            {
                min = input;
            }

            System.out.print("Masukkan angka (-1 untuk mengakhiri): ");
            input = keyboard.nextInt();
        }

        if (max != -1 && min != -1)
        {
              System.out.println("Nilai maksimum dari rangkaian angka-angka yang
diinput = " + max);
               System.out.println("Nilai minimum dari rangkaian angka-angka yang
diinput = " + min);
        }
    }
}
```

***Output Program (MaxMin.java)***

```
Masukkan angka (-1 untuk mengakhiri): 45
Masukkan angka (-1 untuk mengakhiri): 98
Masukkan angka (-1 untuk mengakhiri): 87
Masukkan angka (-1 untuk mengakhiri): 65
Masukkan angka (-1 untuk mengakhiri): 92
Masukkan angka (-1 untuk mengakhiri): 56
Masukkan angka (-1 untuk mengakhiri): 77
Masukkan angka (-1 untuk mengakhiri): -1
Nilai maksimum dari rangkaian angka-angka yang diinput = 98
Nilai minimum dari rangkaian angka-angka yang diinput = 45
```

## 5.5 *Loop* Tersarang

*Loop* yang berada di dalam sebuah *loop* disebut sebagai *loop* tersarang. *Loop* tersarang diperlukan ketika sebuah tugas melakukan operasi berulang dan tugas tersebut juga harus berulang. Salah satu contoh penggunaan *loop* tersarang adalah ketika kita membuat program untuk mencetak sebuah tabel. Misalkan, kita ingin membuat program yang mencetak tabel pemangkatan `x`, `x^2`, `x^3`, dan `x^4` dengan `x = 1, 2, ..., 10` seperti terlihat pada gambar berikut:

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 2 | 4 | 8 | 16 |
| 3 | 9 | 27 | 81 |
| 4 | 16 | 64 | 256 |
| 5 | 25 | 125 | 625 |
| 6 | 36 | 216 | 1296 |
| 7 | 49 | 343 | 2401 |
| 8 | 64 | 512 | 4096 |
| 9 | 81 | 729 | 6561 |
| 10 | 100 | 1000 | 10000 |

Pada gambar di atas, kolom pertama adalah nilai `x`, kolom kedua adalah nilai `x^2`, kolom ketiga adalah nilai `x^3`, dan kolom keempat adalah nilai `x^4`.

*Pseducode* untuk mencetak tabel pemangkatan di atas dapat dituliskan seperti berikut:

```
Untuk x dari 1 sampai 10:
    Cetak baris tabel
    Cetak baris baru
```

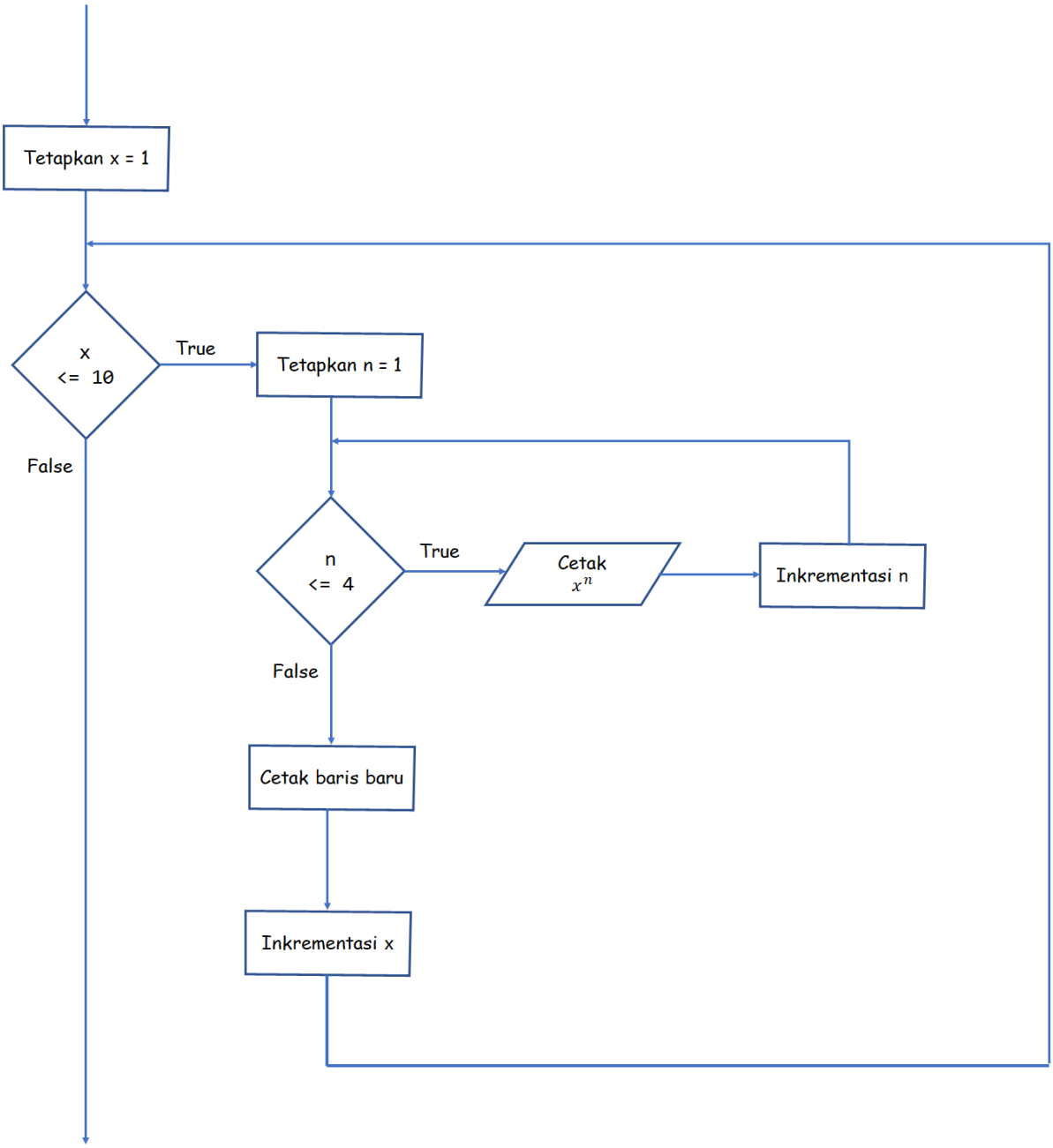
Bagaimana kita mencetak baris tabel? Kita perlu untuk mencetak nilai untuk `x`, `x^2`, `x^3`, dan `x^4`. Ini membutuhkan *loop* kedua dengan *pseducode* seperti berikut:

```
Untuk x dari 1 sampai 4:
    Cetak x^n
```

Sehingga *pseudocode* lengkap menjadi:

```
Untuk x dari 1 sampai 10:
    Untuk n dari 1 sampai 4:
        Cetak x^n
    Cetak baris baru
```

*Flowchart* dari *pseudocode* di atas dapat digambarkan seperti berikut:




Program berikut mengimplementasikan *pseducode* di atas:

***Program (TabelPangkat.java)***

```
/*
    Program ini mencetak tabel pemangkatan x, x^2, x^3, dan x^4
    untuk x dari 1 sampai dengan 10.
*/
public class TabelPangkat
{
    public static void main(String[] args)
    {
        final int NMAX = 4;
        final int XMAX = 10;

        // Loop tersarang untuk mencetak tabel pemangkatan
        for (int x = 1; x <= XMAX; x++)
        {
            for (int n = 1; n <= NMAX; n++)
```

```
            {
                double pangkat = Math.pow(x, n);
                System.out.printf("%10.0f", pangkat);
            }
            System.out.println();
        }
    }
}
```

***Output Program (TabelPangkat.java)***

```
         1         1         1         1
         2         4         8        16
         3         9        27        81
         4        16        64       256
         5        25       125       625
         6        36       216      1296
         7        49       343      2401
         8        64       512      4096
         9        81       729      6561
        10       100      1000     10000
```

Contoh lain dari penerapan *loop* tersarang adalah untuk persoalan mencetak suatu pola. Misalkan, kita membuat sebuah program yang mencetak sebuah pola segitiga dengan karakter `*` seperti berikut:

```
*
**
***
****
```

Pola di atas dapat dicetak menggunakan *loop* tersarang seperti pada program berikut:

***Program (PolaSegitiga.java)***

```
/*
    Program ini mencetak pola segitiga dengan karakter '*'.
*/
public class PolaSegitiga
{
    public static void main(String[] args)
    {
        for (int i = 1; i <=4; i++)
        {
            for (int j = 1; j <= i; j++)
            {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

***Output Program (PolaSegitiga.java)***

```
*
**
***
****
```

# 5.6 Simulasi

Program komputer sering digunakan untuk mensimulasikan kejadian-kejadian di dunia nyata. Program simulasi umum digunakan untuk memprediksi perubahan iklim, menganalisa lalu lintas, memilih saham, dan aplikasi lainnya dalam sains dan bisnis. Program simulasi ini umumnya menggenerasi angka acak dan menggunakan *loop* untuk menggenerasi sejumlah angka acak.

## Menggenerasi Angka Acak

Banyak kejadian di dunia nyata sulit diprediksi dengan akurat, namun kita dapat mengetahui rata-rata perilakunya dengan cukup baik. Sebagai contoh, sebuah toko mungkin mengetahui dari pengalaman bahwa seorang pengunjung datang setiap lima menit. Tentu saja, itu adalah rata-rata. Pengunjung tidak sesungguhnya datang setiap lima menit namun seorang pengunjung datang setiap rata-rata lima menit.

*Class* `Math` mempunyai generator angka acak, `Math.random()`. Memanggil `Math.random()` menghasilkan sebuah angka *floating-point* acak antara >=0 dan <1.

Program berikut melakukan pemanggilan `Math.random()` sebanyak sepuluh kali:

***Program (DemoRandom.java)***

```java
/*
    Program ini mencetak sepuluh angka acak
    antara 0 dan 1.
*/
public class DemoRandom
{
    public static void main(String[] args)
    {
        for (int i = 1; i <= 10; i++)
        {
            double acak = Math.random();
            System.out.println(acak);
        }
    }
}
```

***Output Program (DemoRandom.java)***

```
0.31025333177041536
0.7370112240128699
0.5079289377900715
0.6673275174057173
0.03746424828212003
0.5192316743894589
0.2957961244178555
0.4545042588184789
0.8092926362794155
0.10862488433122419
```

## Mensimulasikan Pelemparan Dadu

Dalam menulis aplikasi simulasi, umumnya kita perlu mengubah *output* dari generator angka acak ke jangkauan yang berbeda. Sebagai contoh, untuk mensimulasikan pelemparan sebuah dadu, kita perlu angka bulat acak antara 1 sampai dengan 6.

Untuk mendapatkan angka bulat acak antara 1 sampai dengan 6 kita dapat menggunakan ekspresi berikut:

```
(int) (Math.random() * 6) + 1
```

Program berikut mensimulasikan pelemparan dua buah dadu.

***Program (Dadu.java)***

```
/*
    Program ini mensimulasikan pelemparan dua buah dadu
    sebanyak sepuluh kali.
*/
public class Dadu
{
    public static void main(String[] args)
    {
        for (int i = 1; i <= 10; i++)
        {
            // Generasi dua angka bulat acak antara 1 s.d 6
            int dadu1 = (int) (Math.random() * 6) + 1;
            int dadu2 = (int) (Math.random() * 6) + 1;
            System.out.println(dadu1 + " " + dadu2);
        }
    }
}
```

***Output Program (Dadu.java)***

*Output 1 (Program dijalankan pertama kali)*

```
3 3
3 5
5 6
3 2
4 4
5 2
2 3
2 6
2 4
4 5
```

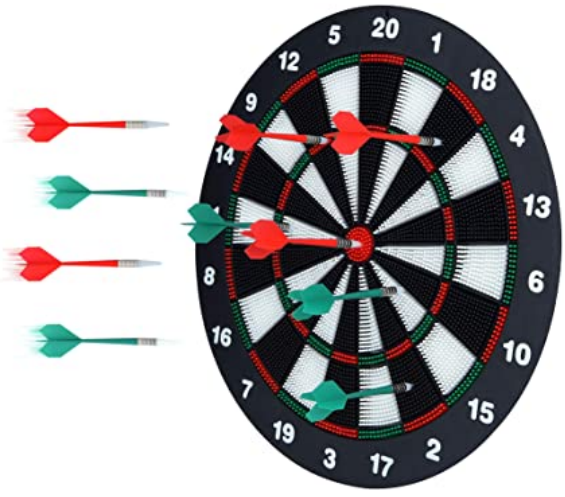
*Output 2 (Program dijalankan kedua kali)*

```
2 6
4 2
6 6
5 3
5 5
4 6
1 4
1 6
3 5
6 2
```

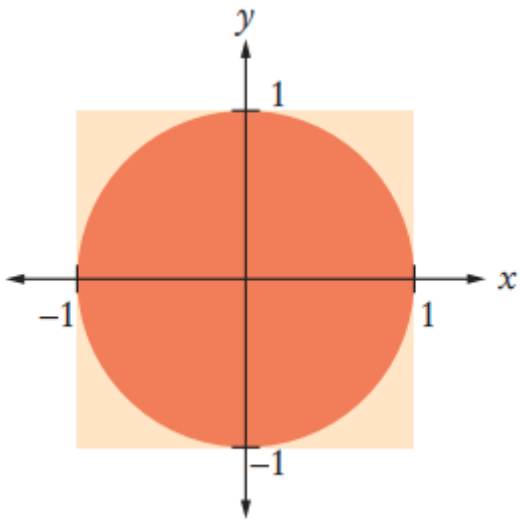
Perhatikan setiap kali kita menjalankan program di atas, kita akan mendapatkan hasil yang berbeda.

## Metode *Monte Carlo*

Metode *Monte Carlo* adalah metode yang umum digunakan untuk mencari solusi aproksimasi dari persoalan yang tidak dapat diselesaikan secara akurat. (Nama Monte Carlo diambil dari nama kasino terkenal di negara *Monaco*.) Sebagai contoh, menghitung nilai angka π secara akurat sangatlah sulit, tapi kita dapat menghitung aproksimasi nilainya dengan metode *Monte Carlo* dengan mensimulasikan pelemparan *dart*.




Untuk mendapatkan nilai aproksimasi dari π, kita mensimulasikan pelemparan *dart* ke sebuah bujur sangkar yang mengelilingi sebuah lingkaran dengan radius 1 seperti pada gambar berikut.




Kita dapat mensimulasikan pelemparan *dart* ini dengan menggenerasi angka acak koordinat `x` dan `y` antara -1 dan 1. Jika angka yang digenerasi berada di dalam lingkaran, yaitu `x^2` + `y^2` ≤ `1`, kita menghitungnya sebagai sebuah *kena*. Karena pelemparan dilakukan secara acak, kita dapat menyimpulkan rasio dari *kena* / *pelemparan* secara aproksimasi sama dengan rasio dari luas lingkaran dan luas bujur sangkar, yaitu π/4. (Luas lingkaran adalah π dan luas bujur sangkar adalah 4). Sehingga, estimasi dari π adalah 4 x *kena* / *pelemparan*. Metode ini akan menghasilkan estimasi dari nilai π.

Untuk menggenerasi angka acak *floating-point* antara -1 dan 1 kita menggunakan:

```
double acak = Math.random();      // 0 <= acak < 1
double x = -1 + 2 * acak;         // -1 <= x < 1
```

Karena nilai `acak` berada di antara 0 (inklusif) sampai dengan 1 (eksklusif), nilai `x` berada di antara -1 + 2 x 0 = -1 (inklusif) sampai dengan -1 + 2 x 1 = 1.

Program berikut menjalankan simulasi ini:

***Program (MonteCarlo.java)***

```
/*
    Program ini menghitung estimasi dari pi dengan mensimulasikan
    pelemparan dart ke bujur sangkar.
*/
public class MonteCarlo
{
    public static void main(String[] args)
    {
          final int PELEMPARAN = 10000;   // Banyaknya pelemparan yang dilakukan
10000 kali

         int kena = 0;    // Untuk menyimpan pelemparan dart yang "kena" (berada
di dalam lingkaran)

        for (int i = 1; i <= PELEMPARAN; i++)
        {
            // Generasi dua angka acak (x dan y) antara -1 dan 1
            double acak = Math.random();
            double x = -1 + 2 * acak;   // koordinat x
            acak = Math.random();
```

```
            double y = -1 + 2 * acak;   // koordinat y

            // Jika x^2 + y^2 <= 1, maka kena
            if (x * x + y * y <= 1)
            {
                kena++;
            }
        }

        // Rasio dari kena / pelemparan secara aproksimasi sama dengan rasio
        // luas lingkaran / luas bujur sangkar = pi / 4
        double estimasiPi = 4.0 * kena / PELEMPARAN;
        System.out.println("Estimasi dari pi: " + estimasiPi);
    }
}
```

***Output Program (MonteCarlo.java)***

```
Estimasi dari pi: 3.1404
```

**REFERENSI**

[1] Horstmann, Cay S. 2012. *Big Java: Late Objects, 1st Edition*. United States of America: John Wiley & Sons, Inc.

[2] Gaddis, Tony. 2016. *Starting Out with Java: From Control Structures through Objects (6th Edition)*. Boston: Pearson.