

# **Operator, Conditional & Looping**

## **Silver - Chapter 1 - Topic 4**

**Selamat datang di Chapter 1 Topic 4  
online course Back End Java dari  
Binar Academy!**



### Welcome Back! 😊

Pada topik sebelumnya kamu udah belajar tentang Java Introduction, Data Types dan Variable.

Pada topik ini, kita bakal mengelaborasi tentang **Operator, Conditional & Looping.** Masih bakal ngomongin tentang tipe data dan kenalan sama istilah-istilah baru.

Yuk kita geser slide-nya~



**Detailnya, kita bakal bahas hal-hal berikut ini:**

- Tipe Reference Data pada Java
- Konsep dasar Array
- Operasi dasar matematis dan perbandingan
- Klasifikasi Expression, Statement dan Block
- Penggunaan If else dan Ternary operation
- Logic flow dari Conditional
- Logic flow dari Looping



Biar makin satset, langsung aja kita mulai yang pertama, nih.

Selamat datang di **String & Reference Data Type~**



Karena sebelumnya udah kenalan sama Primitive Data Type, di topic ini kita bakal kenalan sama Non Primitive Data Types~

Tipe data non primitive atau data reference adalah tipe data yang **nggak didefinisikan oleh Java secara default.**

“Terus, yang mendefinisikan itu siapa?”

**Ya si programmer itu sendiri.**



### Emang fungsinya buat apa, sih?

Tipe data reference dipake buat **menyimpan beberapa value dan punya method-method untuk memanipulasi data**.

Selain itu, value dari tipe data ini bisa bernilai null dan kalau data nggak di-assign dengan value tertentu, maka default value-nya adalah null.



### Berawal dari Reference Data Type~

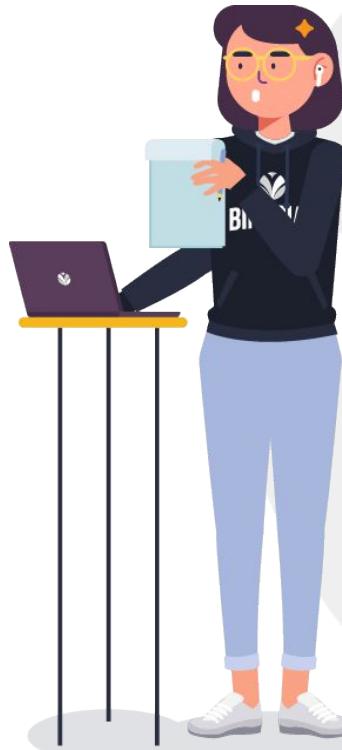
Sebenarnya, sebuah class yang mau dibuat jadi object bakal jadi tipe data reference. Tapiii, hal ini bakal kita pelajari di materi OOP di chapter selanjutnya.

Sekarang, kita bakal bahas nama-nama di tipe data reference yang selalu menggunakan huruf kapital. Contohnya:

int => Integer

long => Long

boolean => Boolean



## Karakteristik Tipe Data String

Tipe data String adalah bagian dari Reference Data Type.

Jadi, tipe data ini **bisa menampung banyak karakter sekaligus**, bisa gabungan huruf, angka, whitespace (spasi), dan berbagai karakter.

Contohnya bisa berupa “aku”, “kamu”, atau “dia”.



### “Terus, cara pakainya gimana?”

Buat gabungin string, kita bisa pake operasi berupa **concatenation**. Salah satunya, yaitu dengan menambahkan operator plus (+).

Contohnya, kamu mau gabungin string “sayang” dengan “kamu”. Maka, kamu tinggal tambahin aja operator +

**"sayang" + "kamu" = "sayangkamu"**

Gampang banget, kan?



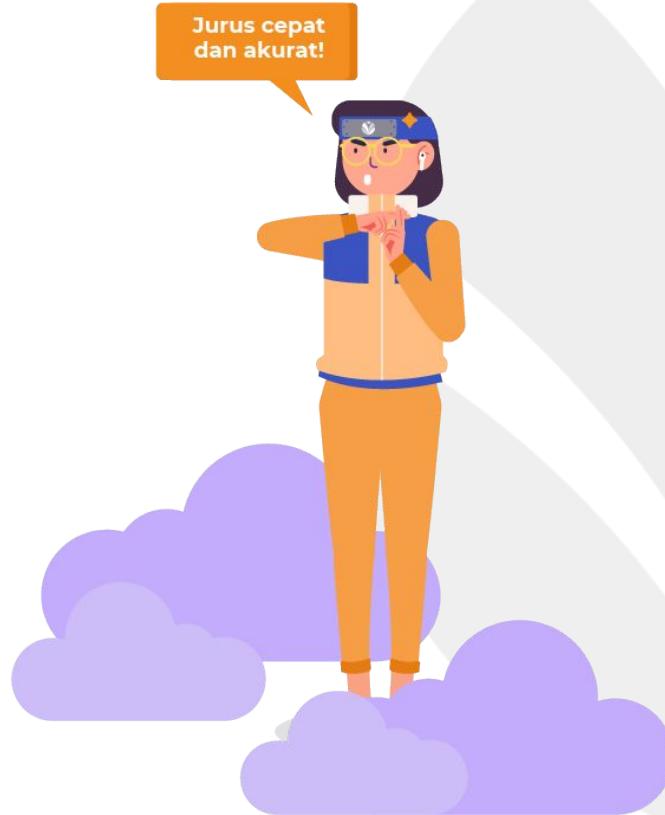
### Ada juga nih trik buat manipulasi String, coba deh!

Trik manipulasi ini bisa dibilang adalah cara cepatnya.  
Supaya praktis~

```
"hello".charAt (0) = 'h'
```

```
"hello".length() = 5
```

```
"hello".indexOf('e') = 1
```



Setelah membahas Non Primitive Data Type dan String, pembahasan kita bakal makin detail, nih.

Ada materi **Array** yang menunggu untuk dijemput kamu.



### Yang jauh mendekat, yang dekat merapat~

Sama kayak kamu kalau lagi nongkrong, Array juga suka ngumpul-ngumpul.

Iya, jadi **Array** itu merupakan sekumpulan data dengan tipe data sama kayak yang disimpan di dalam **memori**, dan masih satu variable.



Nggak cuma suka ngumpul, kumpulan data Array disimpan secara teratur dengan menggunakan index.

Ibarat beli kain di pasar, panjang dari sebuah Array pas udah dideklarasikan nggak bisa ditambah atau dikurangi lagi.

Yang bisa berubah cuma value dari masing-masing indeks.



## “Terus, Array fungsinya buat apa?”

Okey, sekarang kita pake contoh yaaaa~

Misalnya kamu mau menyimpan nama-nama bestie dalam variable. Bisa jadi kamu melakukannya kayak gini:

```
String namaTeman1 = "Sabrina";  
  
String namaTeman2 = "Asep";  
  
String namaTeman3 = "Ucok";  
  
String namaTeman4 = "Nassar";  
  
String namaTeman5 = "Jarjit";
```



**Kalau kamu bikin kayak gitu,  
sah-sah saja sih...**

Tapiii.. gini.

Masalahnya, kalau datanya banyak, misalkan ada 100 data. Gimana? Pasti capek dong bikin variabel sebanyak itu?

Biar nggak cape, kita bisa menyimpan data itu semua dalam Array.



Kayak gini, nih. Berikut adalah contoh dari Array untuk tipe data integer:

- deklarasi array berkapasitas 3 elemen

```
int y[] = new int[3];
```

- deklarasi dan inisialisasi

```
int x[] = {100, 200, 300};
```

- membaca elemen pertama dan ketiga

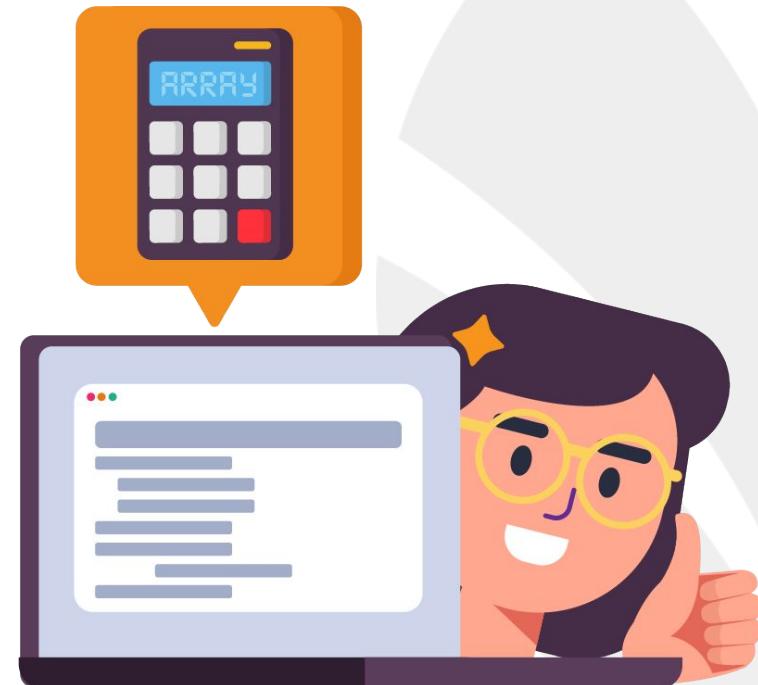
```
int a = x[0] + x[2];
```

- menulis ke elemen kedua

```
x[1] = a + 100;
```

- banyaknya elemen array (3)

```
int numElemen = x.length;
```

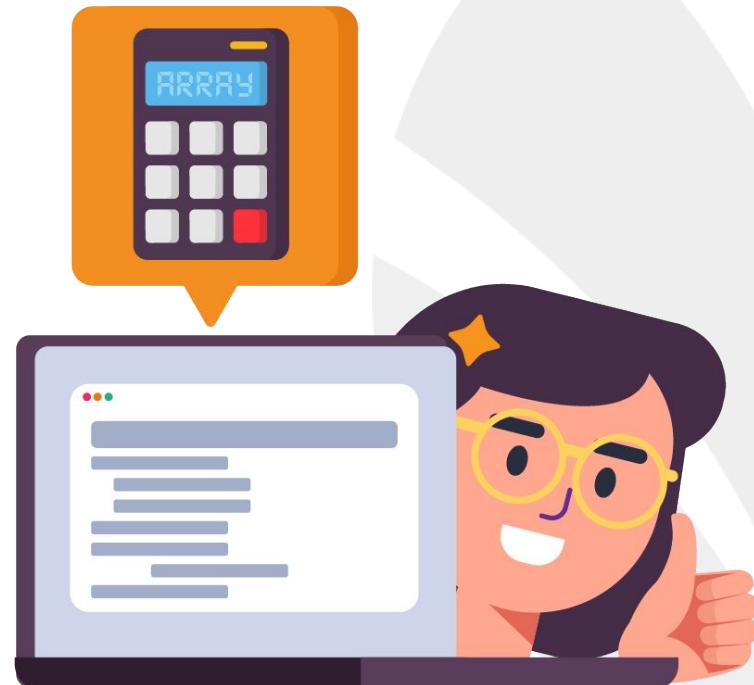


Nah, kalau di bawah ini adalah penerapan dari halaman sebelumnya, yaitu contoh penggunaan array untuk penjumlahan.

```
int x[] = {100,110,113};  
int y[] = x;  
x[0] = x[0] + 200;
```

Dengan melihat pada contoh berikut, maka

- `System.out.println(x[0])` → outputnya 300
- `System.out.println(y[0])` → outputnya 300



### Math, Equality dan Boolean operators~

“Kayak pernah denger materinya deh.  
Tapi lupa, ada di mana, ya?”

Okeyy, kalau gitu kita bahas satu persatu,  
ya!

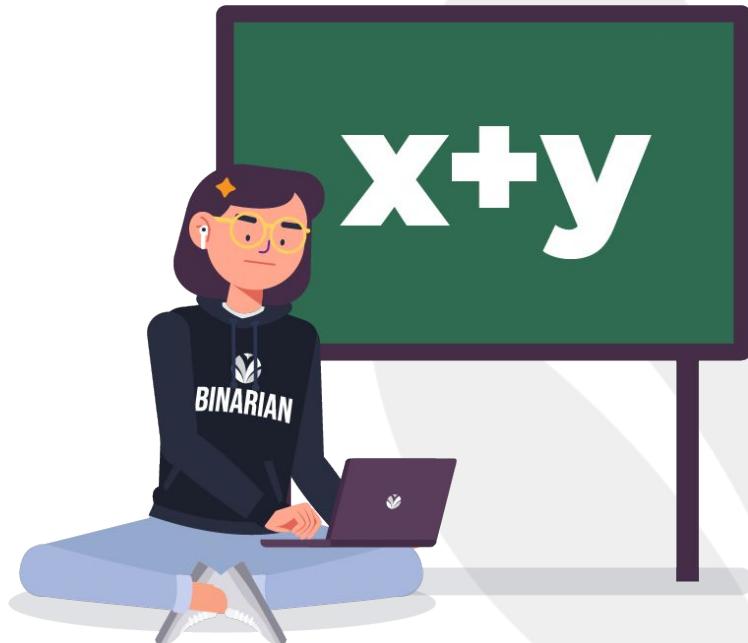


### Kita awali dengan konsep operator dulu~

Eits, ini bukan operator yang suka mengurus mama minta pulsa, ya!

Operator dalam pemrograman digunakan untuk melakukan operasi tertentu.

Misalnya, kita mau menjumlahkan nilai dari variabel x dan y, maka kita bisa pakai operator penjumlahan (+).



### Berikut 6 jenis kelompok operator pemrograman Java!

1. **Binary operator**
2. **Unary operator**
3. **Assignment operator**

(operator sama dengan atau =)

1. **Compound Assignment operator**
2. **Comparison operator**
3. **Logical operator**

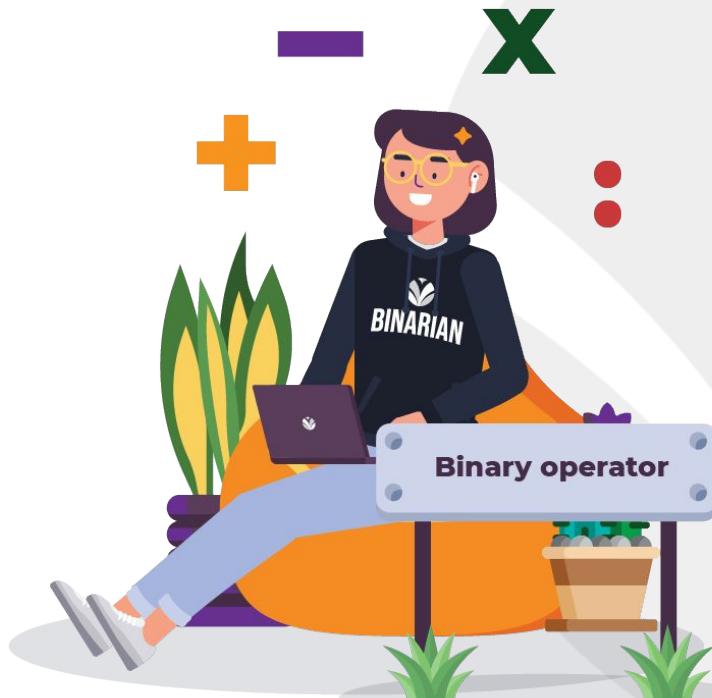
Kita jelaskan semuanya, ya!



## 1. Binary operator

Mirip dengan materi di pelajaran matematika, Binary operator dipakai buat melakukan operasi aritmatika, isinya kayak gini, nih:

- Penjumlahan
- Pengurangan
- Perkalian
- Pembagian
- Sisa (modulus)



**Biar makin kebayang, berikut penjelasannya ya~**

Arti Operator	Operator	Contoh Pemakaian	Keterangan
Penjumlahan	+	sum=num1 + num2	
Pengurangan	-	diff=num1 - num2	
Perkalian	*	prod=num1 * num2	
Pembagian	/	quot=num1 / num2	jika num1 dan num2 adalah integer, pembagian akan menghasilkan nilai integer tanpa mengikutsertakan sisa, jika terdapat sisa.
Sisa (modulus)	%	mod=num1 % num2	Hasil operasi modulus adalah sisa dari operasi num1 / num2. Hasil operasi modulus memiliki tanda ( +/- ) yang sama dengan operand pertama

## 2. Unary operator

Apa sih Unary operator itu?

Jadi, Unary operator adalah operator yang melibatkan satu buah operand.

Misalnya, variable A bernilai negatif 8. Dari situ kamu bisa menggunakan operator negatif sebagai penanda bahwa variable A bernilai negatif ( $-A$ ).

Gampang, kan?



Ini penjelasan Unary operator.

Disimak baik-baik ya~

Arti Operator	Operator	Contoh Pemakaian
Pre-Increment	++operand	int i = 8; int j = ++i; i bernilai 9, j bernilai 9
Post-Increment	operand++	int i = 8; int j = i++; i bernilai 9, j bernilai 8
Pre-Decrement	--operand	int i = 8; int j = --i; i bernilai 7 , j bernilai 7
Post-Increment	operand--	int i=8; int j = i--; i bernilai 7, j bernilai 8

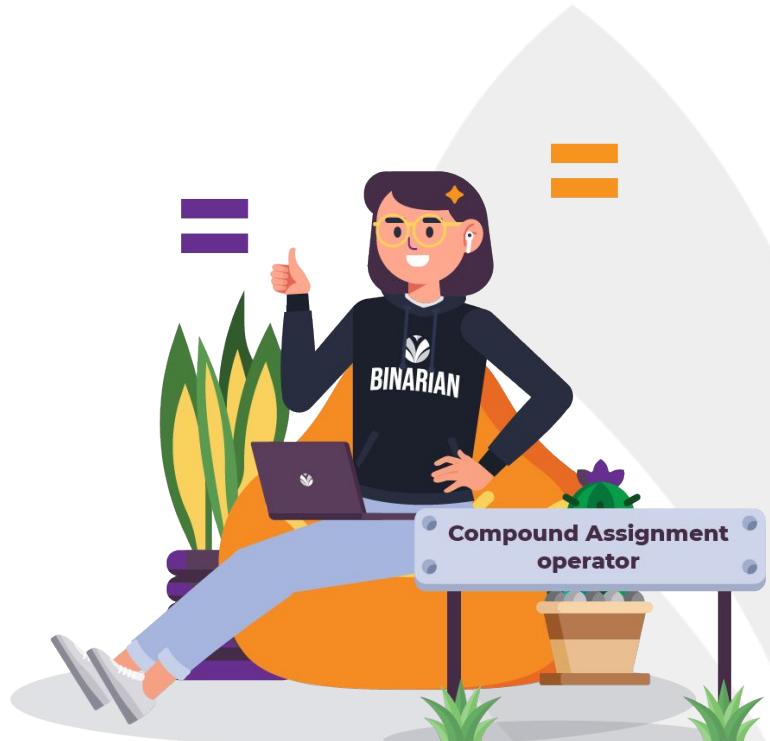
### 3. Compound Assignment operator

Compound assignment operator adalah gabungan dari assignment operator dan binary operator.

Masih inget assignment operator, kan?

Yup, assignment operator adalah operator sama dengan atau =

Belum terlalu paham? Okey, kita next slide ya!



Kalau udah dibaca, pasti paham deh~

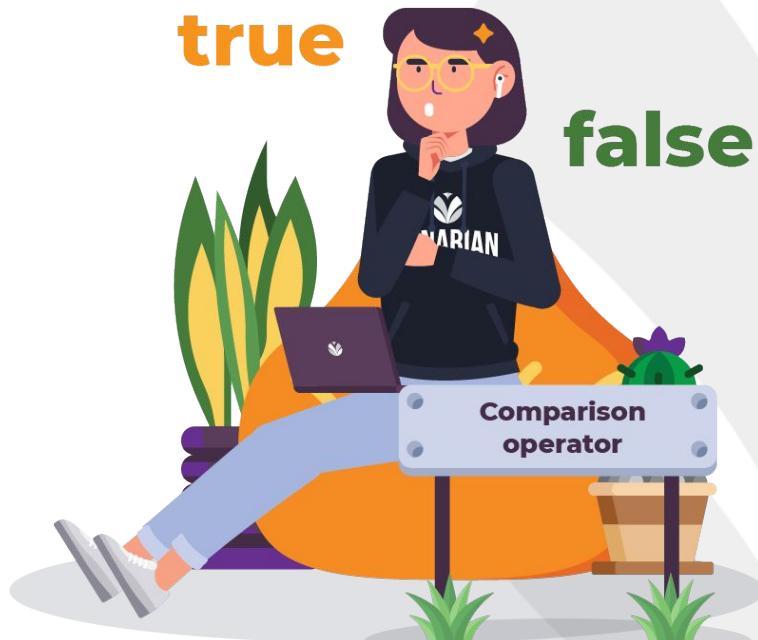
Operator	Nama	Contoh	Equivalent
<code>+=</code>	Addition assignment	<code>i+=5;</code>	<code>i=i+5</code>
<code>-=</code>	Subtraction assignment	<code>j-=10;</code>	<code>j=j-10;</code>
<code>*=</code>	Multiplication assignment	<code>k*=2;</code>	<code>k=k*2;</code>
<code>/=</code>	Division assignment	<code>x/=10;</code>	<code>x=x/10;</code>
<code>%=</code>	Remainder assignment	<code>a%=4;</code>	<code>a=a%4;</code>

## 4. Comparison operator

Yup, sama kayak namanya, jadi tugas operator ini adalah untuk membandingkan.

Beda sama manusia yang nggak suka kalau dibanding-bandingin 😞

Kembali ke laptop, nilai yang dihasilkan dari operator ini berupa boolean, yaitu: true dan false.



### Berikut adalah contoh penggunaannya, gengs!

Condition	Operator	Example
Is equal to (atau “is the same as”)	==	int i = 1; System.out.println(i==1); // (output : true)
Is not equal to (atau “is not the same as”)	!=	int i = 1; System.out.println(i!=1); // (output : false)
Is less than	<	int i = 1; System.out.println(i<1); // (output : false)
Is less than or equal to	<=	int i = 1; System.out.println(i<=1); // (output : true)
Is greater than	>	int i = 1; System.out.println(i>1); // (output : false)
Is greater than or equal to	>=	int i = 1; System.out.println(i>=1); // (output : true)

## 5. Logical operator

Percaya deh, kalau kamu pernah belajar logika matematika, pasti nggak bakal asing sama operator ini.

Logical operator digunakan untuk membuat operasi logika, contohnya:

- Pernyataan 1: Sabrina seorang Back End Engineer.
- Pernyataan 2: Sabrina menggunakan Java.

Kalau ditanya, apakah Sabrina Back End Engineer yang menggunakan Java? Kita perlu cek dulu kebenaran logikanya~



**Beli rumput di toko bunga  
Berikut adalah contohnya~**

Logic	Operator	Contoh
AND	&&	int i = 1; int j = 2; System.out.println((i<1)&&(j>0)); // (output : false)
OR		int i = 1; int j = 2; System.out.println((i<1)  (j>0)); // (output : true)
NOT	!	int i = 1; System.out.println(!(i<3)); // (output : false)

Setelah melewati samudera operator, kita akan ketemu sama tiga best friend 4ever yang saling melengkapi.

Ada si Expression, Statement dan Block yang menunggu kamu!



### Emangnya apa hubungan mereka?

Antara Expression, Statement, dan Block itu memiliki kemampuan mengidentifikasi yang canggih.

Dengan kemampuan mengidentifikasinya si Expression, Statement dan Block, kita bisa lebih gampang buat mengidentifikasi kesalahan kalau terjadi suatu bug.

Nggak cuma itu, kita bahkan bisa meningkatkan ketelitian dalam penulisan code di Java.



**Biar makin paham, simak baik-baik,  
ya!**

Pertama ada Statement. Merupakan **1 baris kode yang diakhiri dengan semicolon atau titik koma (;)**.

Statement juga merupakan unit sintaks pada bahasa pemrograman yang menyatakan aksi atau tugas buat dilakuin.



Gampangnya, statement adalah kode yang berisi perintah. Contohnya kayak gini, nih:

```
System.out.print("Enter an integer: ");
```



## Hubungannya sama expression, apa?

Lanjut nih ke yang berikutnya. **Expression adalah komponen dari statement.**

Contohnya gini, nih

```
System.out.print("Enter an integer: ");
```

- `System.out.print()` // merupakan expression
- `"Enter an integer: "` // merupakan expression





Bentar lagi selesai, nih.

Sekarang waktunya kita memasuki materi  
If else dan Ternary operation~

### Jangan salah ambil keputusan~

“Maksudnya?”

Gini, gini.

**If adalah bentuk operasi dari pengambilan keputusan.**

Block code bakal dijalankan apabila kondisi yang divalidasi terpenuhi, sehingga If bisa mengambil keputusan.



## If pada Java juga punya syntax, lho!

Syntax-nya kayak gini, nih:



```
If ( //kondisi yang harus dipenuhi){  
    //block kode yang akan dalankan jika  
    kondisi terpenuhi;  
}
```



```
If ( //kondisi yang harus dipenuhi)  
//statement yg dalankan jika kondisi  
terpenuhi;
```

Ataupun jika block hanya terdiri dari 1 statement maka bisa berbentuk.

### Selanjutnya, ada istilah lain, yaitu Else~

**Else** merupakan block of code yang dijalankan kalau kondisi nggak dipenuhi.

Meskipun Operator If bisa berdiri tanpa else, tapi operator else nggak bisa berdiri tanpa if.

If else pada Java punya syntax seperti gambar di samping



```
If ( //kondisi yang harus dipenuhi){  
    //block kode yang akan jalankan jika  
    kondisi terpenuhi;  
} else{  
    //block kode yang akan jalankan jika  
    kondisi tidak terpenuhi;}
```

## Contoh penggunaan If Else~

Sedangkan ternary operation merupakan operasi if else yang bisa dipakai kalau kedua if-else block kode cuma punya 1 statement.

Berikut contoh ternary operation yang equivalent dengan contoh di atas:

**String hasil = nilai>65 ? “Lulus” : “Gagal”;**



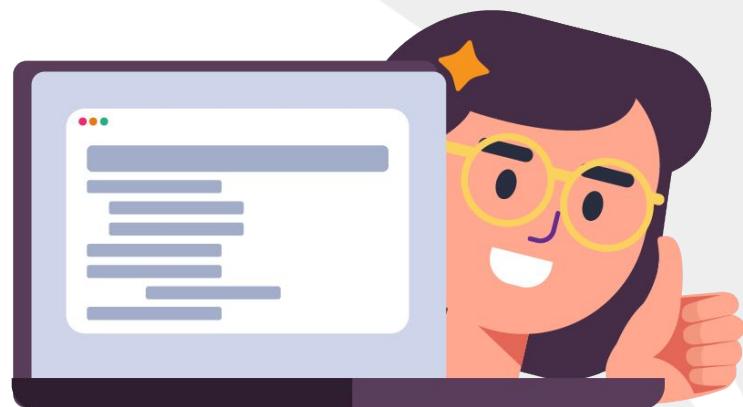
```
String hasil;
if ( nilai > 65 ){
    hasil = "Lulus");
} else{
    hasil = "Gagal";
}
```

### Ada If, ada Else, ada juga Ternary Operation~

Ternary operation merupakan **operasi If Else yang bisa dipake kalau kedua if-else block kode cuma punya 1 statement.**

Berikut contoh ternary operation yang equivalent dengan contoh sebelumnya:

```
String hasil = nilai>65 ? "Lulus" : "Gagal";
```



## If else bisa digunakan dengan lebih dari satu kondisi

Contohnya kayak gambar di samping.

Karena kode di atas punya kondisi yang nggak punya login AND, OR, maka kita bisa pake switch case.



```
String hasil;
if ( nilai.equals("A")){
    hasil = "lulus dengan sempurna");
} else if (nilai.equals("B")){
    hasil = "Lulus dengan baik";
} else if (nilai.equals("C")){
    hasil = "Lulus tapi mepet";
} else{
    hasil = "Tidak lulus";
}
```

“Switch Case? Apa lagi, tuh?”

Nah, udah penasaran sama Switch Case,  
kan?

Kalau gitu, ready? Go!



### Serupa tapi tak sama, apakah itu?

Switch merupakan bentuk pengambilan keputusan dengan memvalidasi seluruh kondisi yang mungkin terpenuhi.

**Switch ini hampir mirip dengan pengambilan keputusan If- else if - else**, tapi kondisi yang terpenuhi nggak boleh berbentuk kondisi yang majemuk (punya relasi `&&` dan `||`).



```
String hasil;
Switch(nilai){
    case "A":
        hasil = "Lulus dengan sempurna";
        break;
    case "B":
        hasil = "Lulus dengan baik";
        break;
    case "C":
        hasil = "Lulus tapi mepet";
        break;
    default:
        hasil = "Tidak lulus";
}
```

Berikut contoh penggunaan switch-case dengan padanan contoh If- else if - else!

Penjelasan:

- **switch** adalah kata kunci yang mengindikasikan dimulainya konstruksi switch;
- **variable** adalah variabel yang nilainya bakal dievaluasi. Variable cuma bisa bertipe-data char, byte, short, atau int;

```
String hasil;
Switch(nilai){
    case "A":
        hasil = "Lulus dengan sempurna";
        break;
    case "B":
        hasil = "Lulus dengan baik";
        break;
    case "C":
        hasil = "Lulus tapi mepet";
        break;
    default:
        hasil = "Tidak lulus";
}
```

## Lanjuttt...

Penjelasan:

- **case** adalah kata kunci yang mengindikasikan sebuah nilai yang diuji. Kombinasi kata kunci case dan nilai\_literal disebut case label;
- **nilai\_literal\_k** adalah nilai yang mungkin bakal jadi nilai variabel. nilai\_literal\_k nggak bisa berupa variabel, ekspresi, atau method, tapi bisa berupa konstanta. k = {default, 1,2,...,n};

```
String hasil;
Switch(nilai){
    case "A":
        hasil = "Lulus dengan sempurna";
        break;
    case "B":
        hasil = "Lulus dengan baik";
        break;
    case "C":
        hasil = "Lulus tapi mepet";
        break;
    default:
        hasil = "Tidak lulus";
}
```

## Penjelasan:

- **break** adalah pernyataan yang sifatnya opsional, yang bikin aliran program keluar dari blok switch.  
Kalau setelah code\_block\_k nggak ada pernyataan break, maka aliran program bakal masuk ke case berikutnya. k = {default, 1,2,3,...,n};
- **default** adalah kata kunci yang mengindikasikan code\_block\_default bakal dieksekusi kalau semua case yang diuji nggak sesuai sama nilai variabel

Udah nggak asing sama kata **For**, kan?

Buat mengawali materi hari ini, kita bakal bahas tentang **For**.

Semangattt~



**Kalau mau menyuruh komputer mengerjakan perintah yang berulang-ulang tuh gimana, ya?**

Misalnya kita mau nyuruh komputer buat menampilkan teks "Sabrina cantik banget" sebanyak 5x.

Kita bisa aja meminta kayak gini,

```
System.out.println("Sabrina cantik banget");  
System.out.println("Sabrina cantik banget");  
System.out.println("Sabrina cantik banget");  
System.out.println("Sabrina cantik banget");  
System.out.println("Sabrina cantik banget");
```



### Tapi...

Gimana kalau kita mau menulis kalimat tersebut sampai 1000 kali, apa kita mampu untuk mengetik kode sebanyak itu?

Tentunya nggak dong, oleh karena itu, kita harus pakai Loop.



### So, Loop itu artinya apa?

Jadi, **Looping** adalah urutan logika yang terus menerus diulang sampai suatu kondisi bisa tercapai.

Kondisi itu bisa berupa pembentukan pola, membuat sesuatu jadi dinamis, atau bisa juga bikin suatu urutan.



Buat bikin perulangan ini, kita harus perhatikan kondisi **stop**.

Kalau kondisi stop nggak tercapai, maka bisa terjadi sebuah **infinite loop** (perulangan yang nggak ada akhirnya) yang mungkin aja bikin berbagai runtime error.

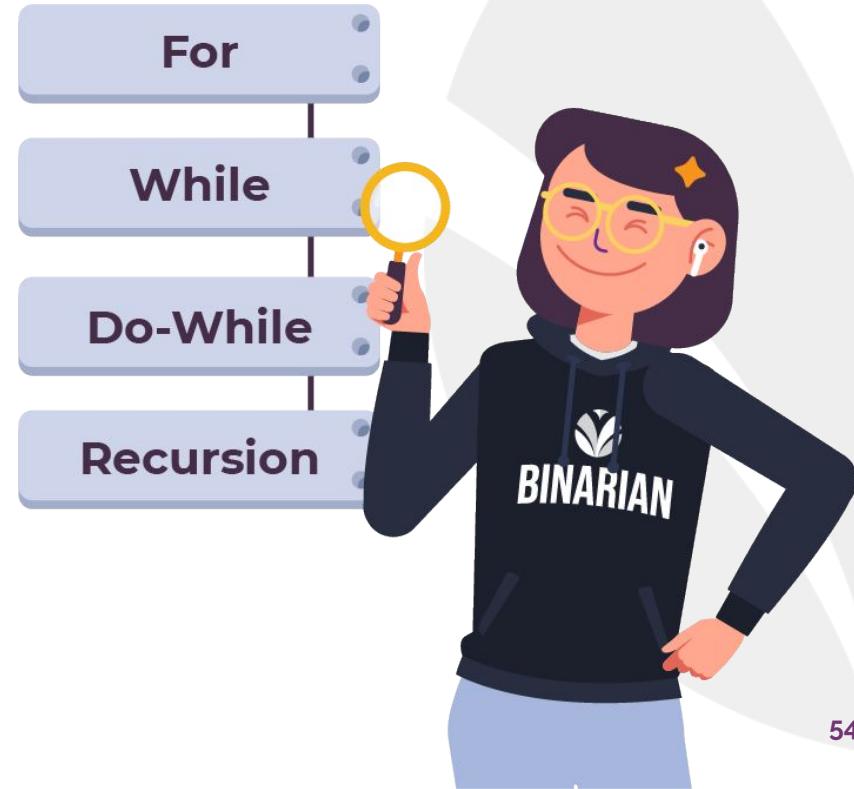


## Looping juga punya metode, lho!

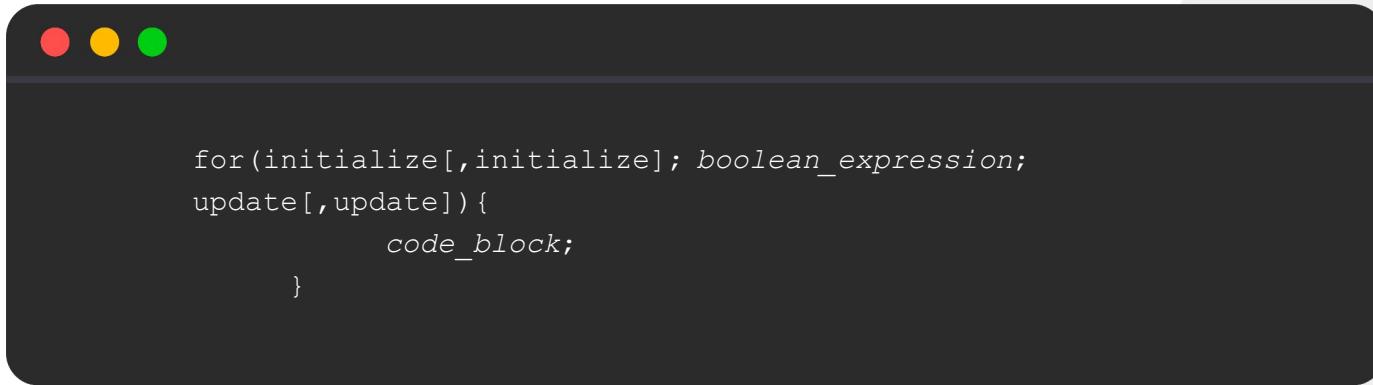
Ada 4 metode buat melakukan looping, yaitu:

1. **For**
2. **While**
3. **Do-While**
4. **Recursion**

Kita bahas satu-satu ya!



### Bentuk umum dari konstruksi Loop menggunakan For

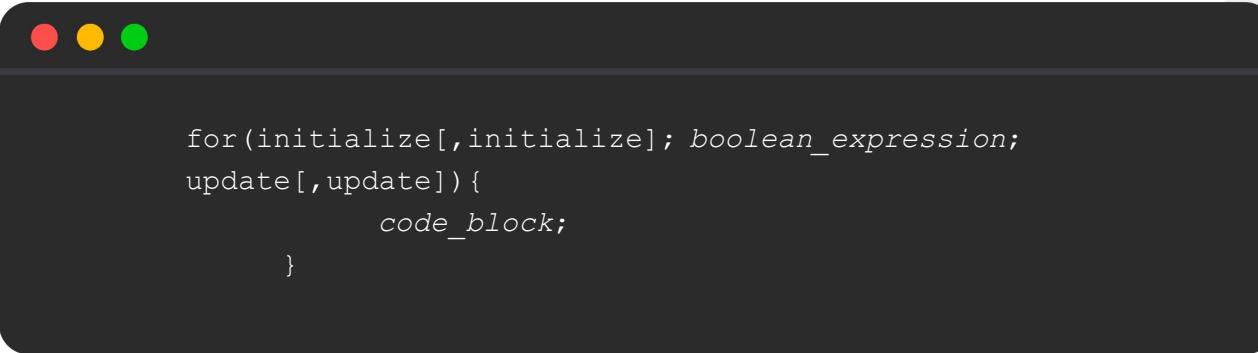
A screenshot of a dark-themed code editor window. At the top, there are three circular window control buttons (red, yellow, green). Below them is a thin horizontal bar. The main area contains the following pseudocode:

```
for(initialize[, initialize]; boolean_expression;  
     update[, update]) {  
    code_block;  
}
```

The code is written in a monospaced font, with variable parts like "initialize", "boolean\_expression", and "update" in bold, italicized black, while the "code\_block" part is in a standard black font.

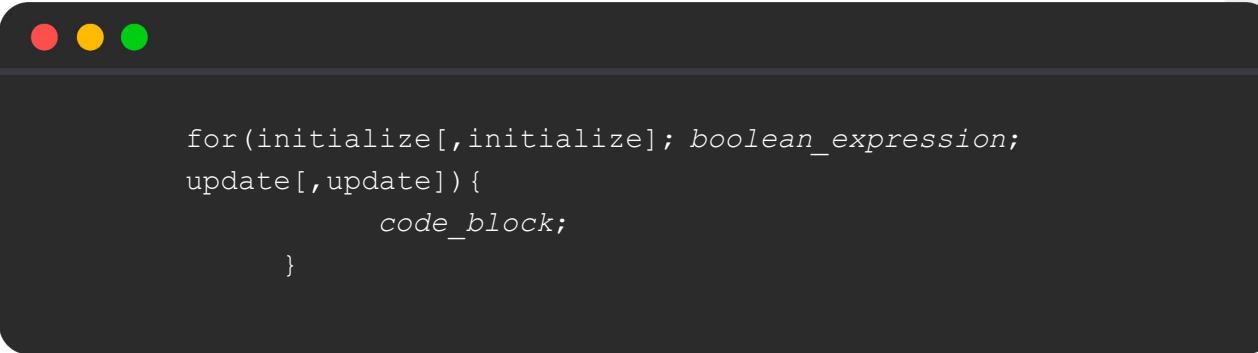
Bagian **initialize**, berisi pernyataan pemberian nilai awal untuk suatu variable parameter;

Parameter ini berfungsi sebagai counter.



Bagian **boolean\_expression** berisi pernyataan logika yang nantinya bakal diperiksa sebagai syarat pengulangan biar terus dilanjutkan.

Pengulangan bakal dilanjut kalau nilai ekspresi Boolean di segmen ini bernilai **true**. Tapi, kalau ekspresi Boolean nggak terpenuhi maka **looping akan stop**.

A screenshot of a Mac OS X application window. The title bar is dark grey with three colored window control buttons (red, yellow, green) on the left. The main area is a light grey text editor containing a template for a for loop:

```
for(initialize[,initialize]; boolean_expression;  
update[,update]) {  
    code_block;  
}
```

The code is written in a monospaced font, with variable parts like "initialize", "boolean\_expression", "update", and "code\_block" in italicized font to indicate they are placeholders.

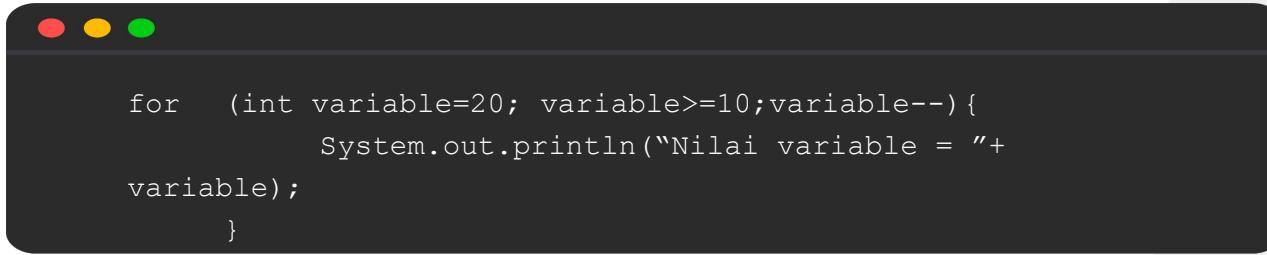
Bagian **update** merupakan pernyataan untuk memperbarui nilai parameter pada satu putaran, ketika loop selesai dieksekusi.

**Ada beberapa catatan yang perlu kamu perhatikan untuk metode looping For, yaitu:**

- Bagian initialize bisa diisi pakai lebih dari 1 pernyataan initialize.
- Bagian boolean\_expression cuma bisa diisi sama 1 pernyataan logika.
- Bagian update bisa diisi pakai lebih dari 1 pernyataan update.



### Ada juga nih contoh dari penerapan for~



```
for (int variable=20; variable>=10;variable--) {
    System.out.println("Nilai variable = "+
variable);
}
```

- **Int variable=20** merupakan bentuk dari initialize, artinya nilai variable awal adalah 20.
- **variable>=10** merupakan bentuk dari Boolean expression. Artinya kalau nilai dari variable < 10, maka loop nggak bakal dilanjutkan.
- **variable--** merupakan bentuk update value yang artinya setiap satu loop selesai, maka nilai variable akan dikurangi 1. Terus nilai variable yang dipakai pada loop selanjutnya bakal pakai nilai variable yang baru.

Coba kamu lihat contoh di bawah ini!

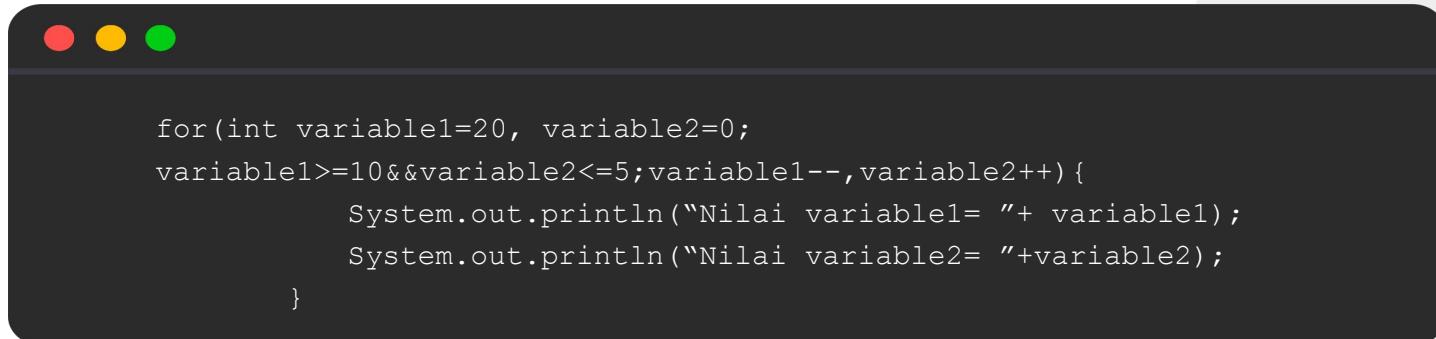
```
for(int variable=20; variable>=10;variable--) {  
    System.out.println("Nilai variable = "+  
variable);  
}
```

Nah, nanti outputnya bakal jadi kayak gini~

Nilai variable = 20  
Nilai variable = 19  
Nilai variable = 18  
Nilai variable = 17  
Nilai variable = 16

Nilai variable = 15  
Nilai variable = 14  
Nilai variable = 13  
Nilai variable = 12  
Nilai variable = 11  
Nilai variable = 10

Kalau yang ini, contoh penggunaan looping For pakai 2 variable~



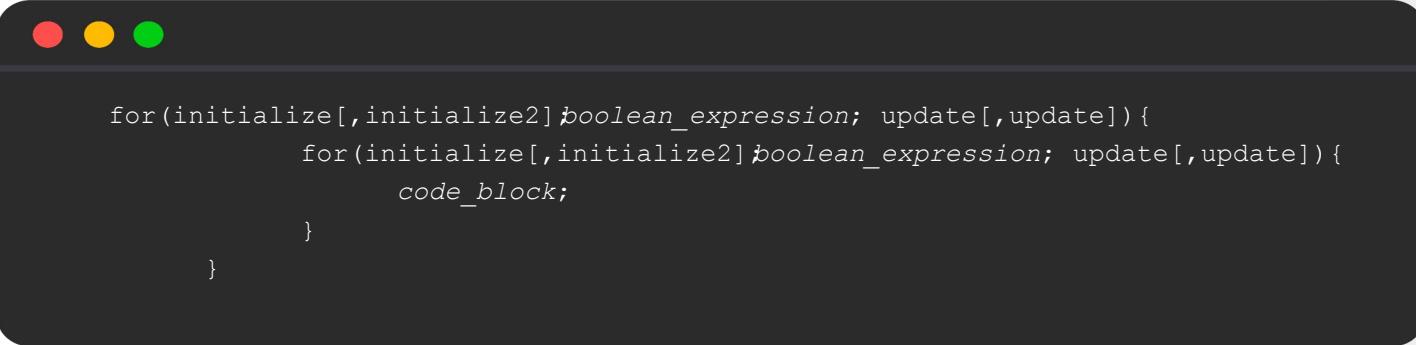
```
for(int variable1=20, variable2=0;  
variable1>=10&&variable2<=5;variable1--,variable2++) {  
    System.out.println("Nilai variable1= "+ variable1);  
    System.out.println("Nilai variable2= "+variable2);  
}
```

Menurut kamu, Outputnya bakal kayak gimana?

### Kondisi penggunaan Loop juga harus kita perhatiin, lho

Ada kondisi di mana kita harus pake loop di dalam sebuah loop. Dimana kondisi looping ini disebut dengan **nested loop** atau perulangan bersarang.

Kalau masih bingung, perhatikan contoh struktur nested loop berikut, ya!



```
for(initialize[,initialize2] boolean_expression; update[,update]) {
    for(initialize[,initialize2] boolean_expression; update[,update]) {
        code_block;
    }
}
```

The image shows a dark-themed code editor window. At the top left, there are three colored circular icons: red, yellow, and green. The main area contains a piece of pseudocode illustrating a nested loop structure. It consists of two nested 'for' loops. The inner loop has an empty body labeled 'code\_block'. Both loops have placeholder parameters for initialization, termination conditions, and updates. The code is presented in a monospaced font.

Dalam hal ini, buat melakukan initialize kita harus bikin 2 parameter.

Contoh dari penggunaan nested loop bisa kamu lihat pada gambar di samping ini, yaaa~

```
int rows = 5;

for (int i = 1; i <= rows; ++i) {

    for (int j = 1; j <= i; ++j) {
        System.out.print(j + " ");
    }
    System.out.println("");
}
```

Nantinya, bakal menghasilkan output kayak gini~

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

Setelah ada gambaran tentang konsep For, selanjutnya ada materi yang tentang While.

Penasaran? Yuk kita pelajari!

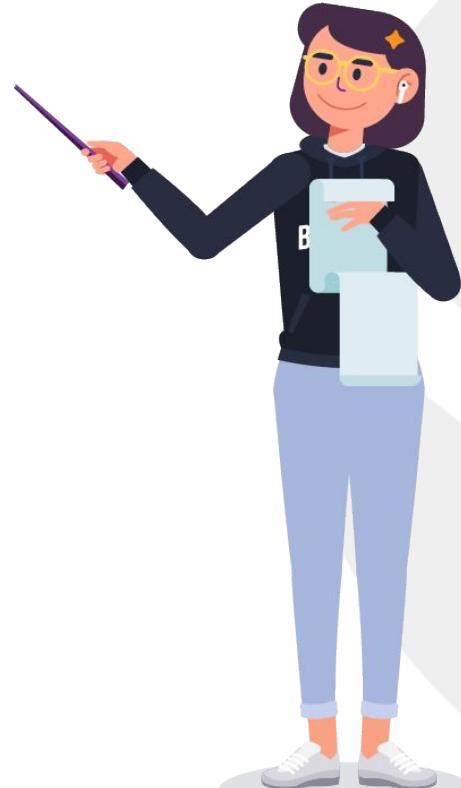


### Yang nggak asing emang selalu bikin penasaran, kan?

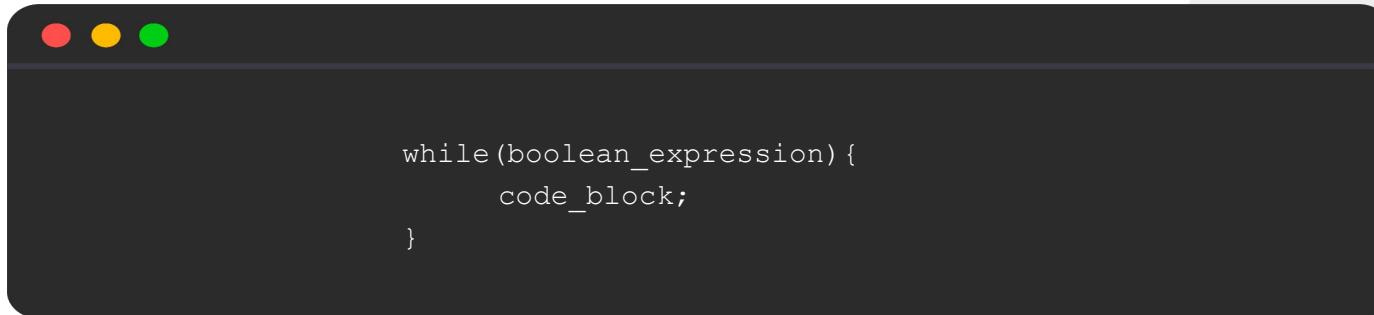
While disini masih berkaitan sama konsep Loop.

**While adalah konstruksi pengulangan yang jumlah loopnya tergantung pada suatu kondisi.**

Kalau pake For kita bisa memperkirakan berapa jumlah loop yang terjadi, sedangkan kalau pake while kita nggak bakal secara eksplisit dijelaskan di notasi while.



### Berikut bentuk umum dari while!



```
while (boolean_expression) {  
    code_block;  
}
```

**boolean\_expression** di atas adalah suatu kondisi di mana ketika kondisi tersebut terpenuhi, maka loop bakal dilanjutin. Kalau kondisi nggal terpenuhi, ya nggak bakal dilanjutin.

Ibarat Cowok yang suka sama cewek, kalau si cewek udah sesuai sama tipe idealnya, pasti si cowok bakal langsung proses PDKT, kan?

Takut salah kalau pakai While?

Berikut contoh yang salah dari penggunaan while!

```
int num1 = 0;
int num3 = 23;
while(num3 > num1) {
    System.out.println("num1=" + num1 + ", num3=" + num3);
}
```

Kira-kira, gimana outputnya?

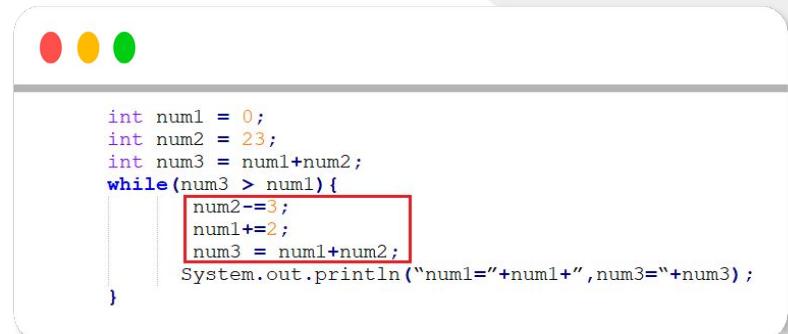
## Lho, apanya yang salah?

Kalau diperhatikan baik-baik, dicontoh sebelumnya **nggak ada kondisi update value**.

Kondisi tersebut yang bakal menyebabkan infinite loop karena kondisi `num3>num1` bakal selalu terpenuhi.

Gambar disamping adalah contoh penggunaan loop while yang benar~

- Bagian yang ditandai, merupakan update value dari variable.
- Statement update value selalu ada di dalam code block while.



A screenshot of a Java code editor window. The code is as follows:

```
int num1 = 0;
int num2 = 23;
int num3 = num1+num2;
while(num3 > num1){
    num2-=3;
    num1+=2;
    num3 = num1+num2;
}
System.out.println("num1="+num1+",num3="+num3);
```

The lines `num2-=3;`, `num1+=2;`, and `num3 = num1+num2;` are highlighted with a red rectangular box, indicating they are the update statements within the loop block.

**Penggunaan loop while yang benar  
di halaman sebelumnya, bakal  
nampilin output sebagai berikut!**

```
num1=2, num3=22
num1=4, num3=21
num1=6, num3=20
num1=8, num3=19
num1=10, num3=18
num1=12, num3=17
num1=14, num3=16
num1=16, num3=16
```

Setelah kondisi `num3 > num1` nggak terpenuhi, maka loop nggak bakal dilanjutkan.

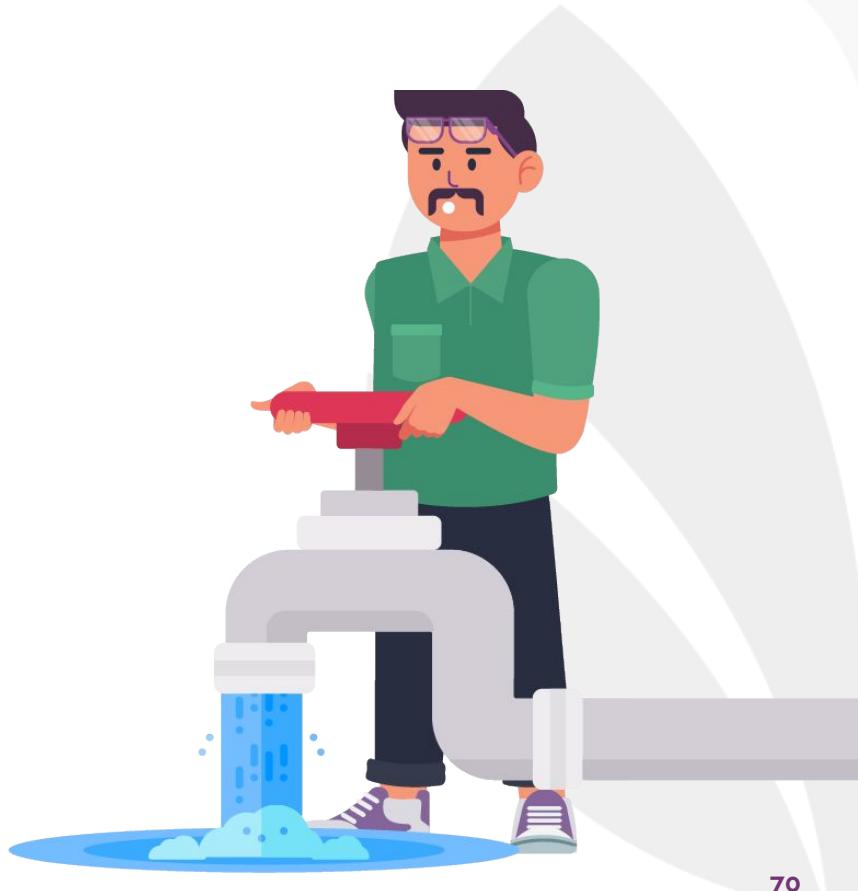


### Jangan lupa exit dari Loop, ya!

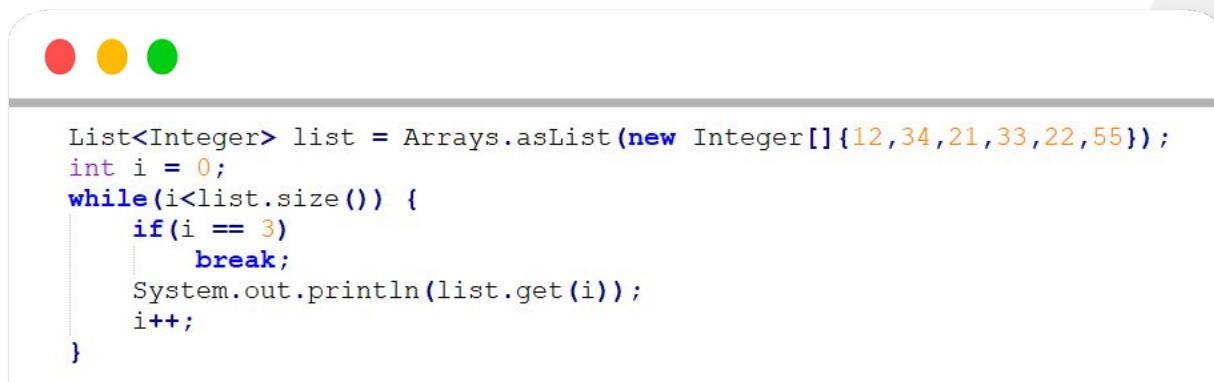
Bukan cuma keran air aja yang harus ditutup kalau udah nggak dipakai, Loop juga sama. Berikut cara exit Loop!

1. Mengikuti kondisi boolean expression pada while seperti yang udah dicontohin sebelumnya.
2. Pakai break.
3. Pakai return.

Exit dengan cara-cara di atas juga berlaku buat metode loop yang lain, lho. Kita bahas satu-satu yuk!



## Exit dengan menggunakan break~

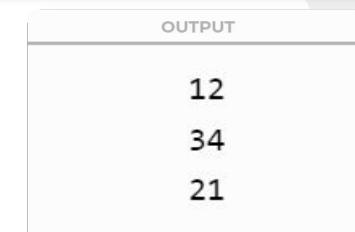


A screenshot of a Java code editor window. The title bar has three colored circles (red, yellow, green). The code in the editor is:

```
List<Integer> list = Arrays.asList(new Integer[]{12,34,21,33,22,55});
int i = 0;
while(i<list.size()) {
    if(i == 3)
        break;
    System.out.println(list.get(i));
    i++;
}
```

Jika variable i bernilai 3 maka loop nggak bakal dilanjutkan.

Berikut outputnya:



A screenshot of a terminal window titled "OUTPUT". The output shows the numbers 12, 34, and 21, each on a new line, indicating that the loop was terminated by the break statement when i reached 3.

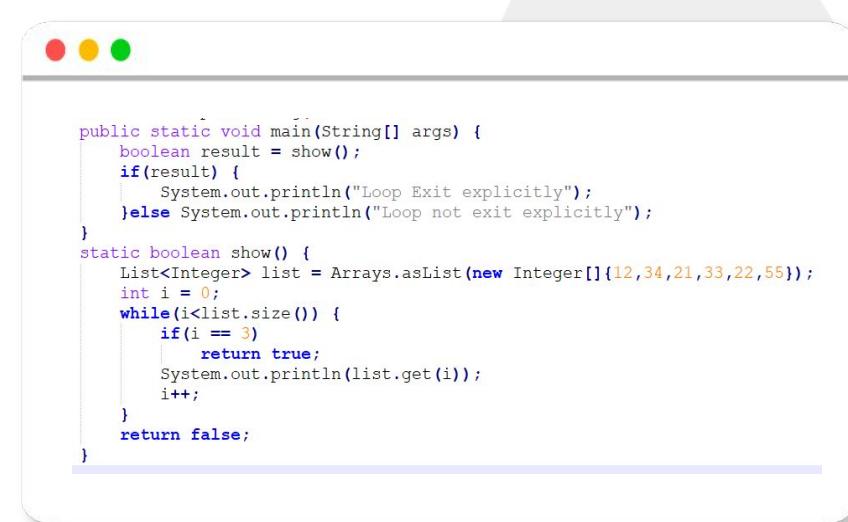
```
12
34
21
```

## Sekarang kita coba exit pake Return, ya

Kalau pakai cara ini, loop while harus ada di suatu method yang bukan main method.

Method tersebut bisa berupa method dengan return ataupun void method.

Disamping ini merupakan contohnya!

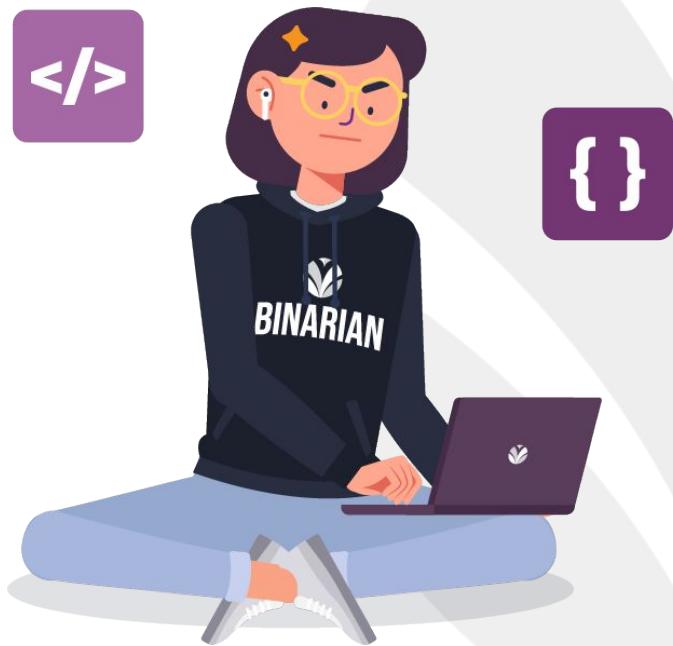


```
public static void main(String[] args) {
    boolean result = show();
    if(result) {
        System.out.println("Loop Exit explicitly");
    } else System.out.println("Loop not exit explicitly");
}
static boolean show() {
    List<Integer> list = Arrays.asList(new Integer[]{12, 34, 21, 33, 22, 55});
    int i = 0;
    while(i<list.size()) {
        if(i == 3)
            return true;
        System.out.println(list.get(i));
        i++;
    }
    return false;
}
```

Pada contoh sebelumnya, kondisi exit loopnya ( $i=3$ ) kalau terpenuhi, maka nilai yang bakal di-return adalah true.

Sehingga outputnya:

OUTPUT
12
34
21
Loop Exit explicitly



Sedangkan yang ini adalah contoh penggunaannya pada void method~



```
public static void main(String[] args) {
    show();
}
static void show() {
    List<Integer> list = Arrays.asList(new Integer[]{12,34,21,33,22,55});
    int i = 0;
    while(i<list.size()) {
        if(i == 3)
            return;
        System.out.println(list.get(i));
        i++;
    }
}
```

Pada statement return;

Nggak mengembalikan nilai apa-apa karena method show adalah void method.

Jadi, outputnya sebagai berikut:

### OUTPUT

12

34

21

For, udah.

While, udah juga.

Berarti sekarang saatnya: **Do-While**.



### “While sama Do-While, apa bedanya?”

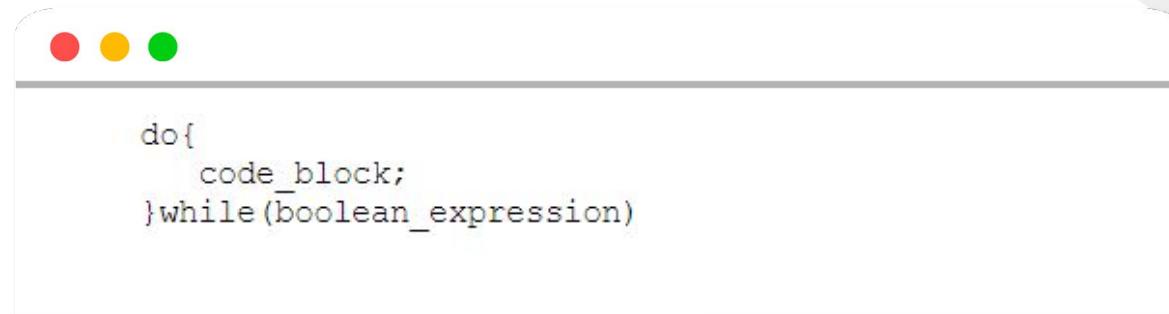
Sebetulnya, looping dengan menggunakan Do-While mirip dengan menggunakan While.

**Bedanya, eksekusi code block di loop dilakukan terlebih dahulu** di metode Do-While.

Setelah mengeksekusi code block dalam loop, baru deh kondisinya di cek. Kalau kondisinya terpenuhi, berarti loop bisa dilanjutkan.



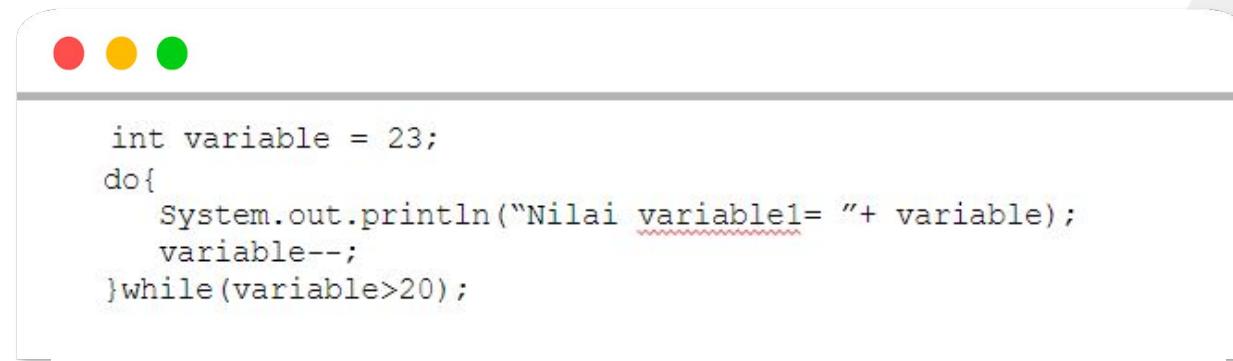
Ini adalah bentuk umum dari Do-While~



```
do{  
    code_block;  
}while(boolean_expression)
```

**code\_block** merupakan block yang dieksekusi duluan. Terus di cek kondisinya sama **boolean\_expression**.

Mau contoh? Nih, contohnya~



```
int variable = 23;
do{
    System.out.println("Nilai variable= "+ variable);
    variable--;
}while(variable>20);
```

Outputnya adalah sebagai berikut:

OUTPUT
23
22
21
20

Kalau outputnya bernilai 20, maka loop nggak bakal dilanjutkan.

Pernah dengar istilah **Recursion**?

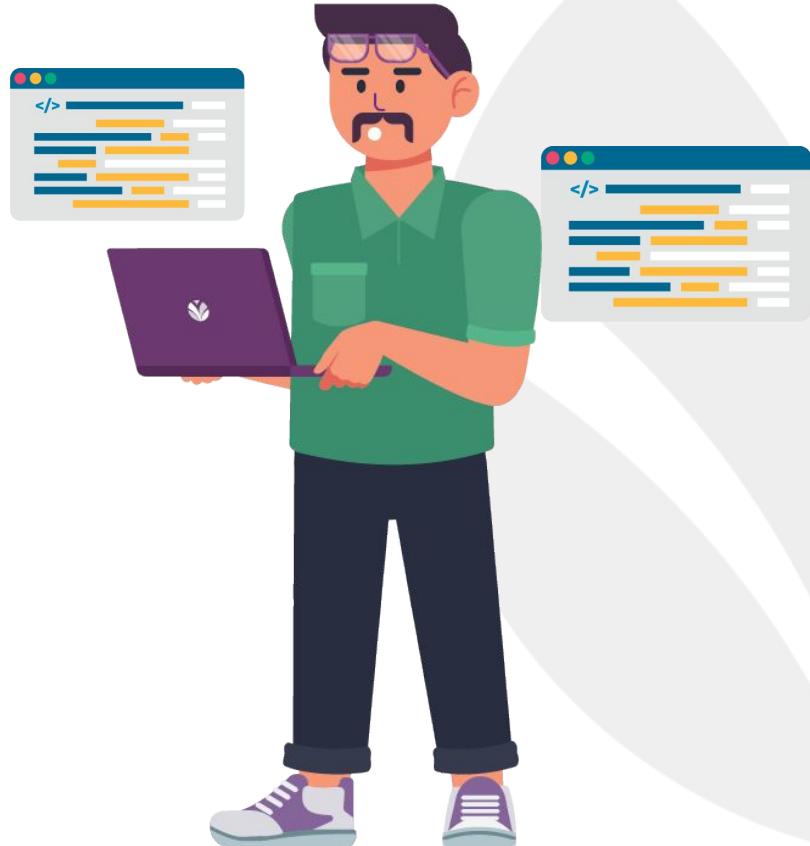
Yup, materi terakhir hari ini bakal kupas tuntas tentang **Recursion**.

Tunggu apa lagi? Ayo kita next slide!



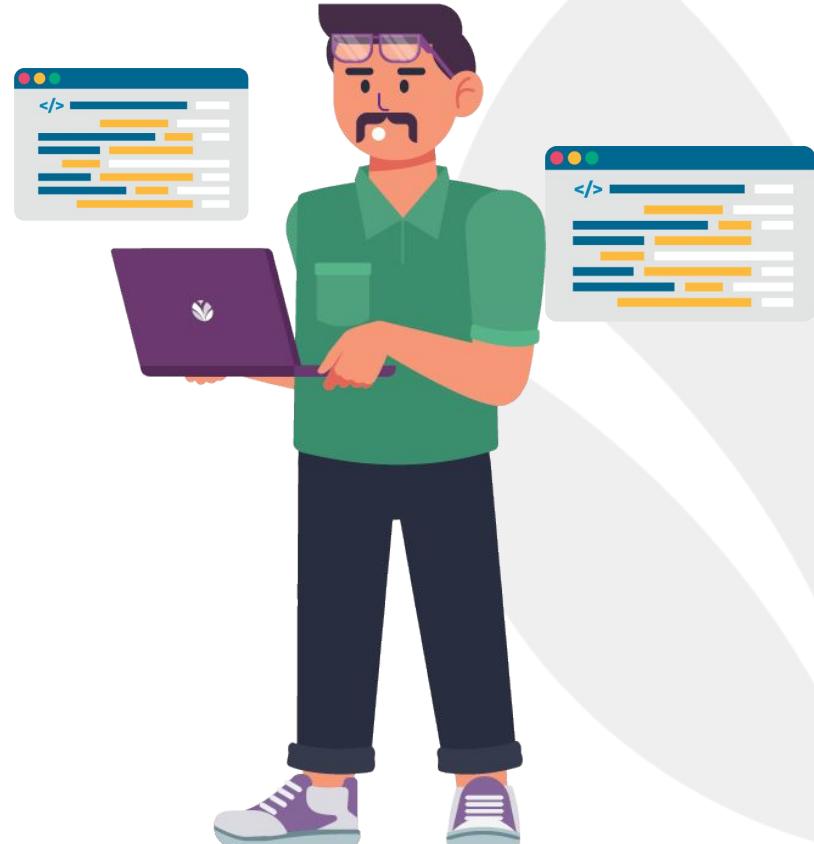
### Recursion itu apa, sih?

**Recursion** adalah bentuk dari loop dengan cara memanggil method di dalam method. Method ini disebut dengan recursive method.



Recursion punya performance yang lebih buruk dibandingkan dengan metode yang lainnya. Tapi, penggunaan recursion bisa menyelesaikan masalah looping yang nggak bisa diselesaikan dengan metode loop lain, seperti while, for dan do-while, lho.

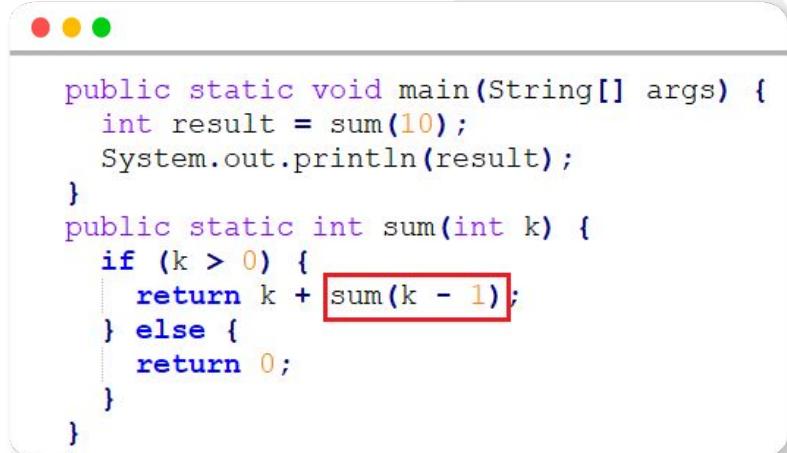
Kecil kecil cabe rawit, ya!



## Nih contohnya, sob~

Di dalam content method sum, method tersebut memanggil dirinya sendiri.

Tapi, supaya recursive method nggak menghasilkan infinite loop, maka pemanggilan method diri sendiri harus berada **di dalam code block** dari stop condition.



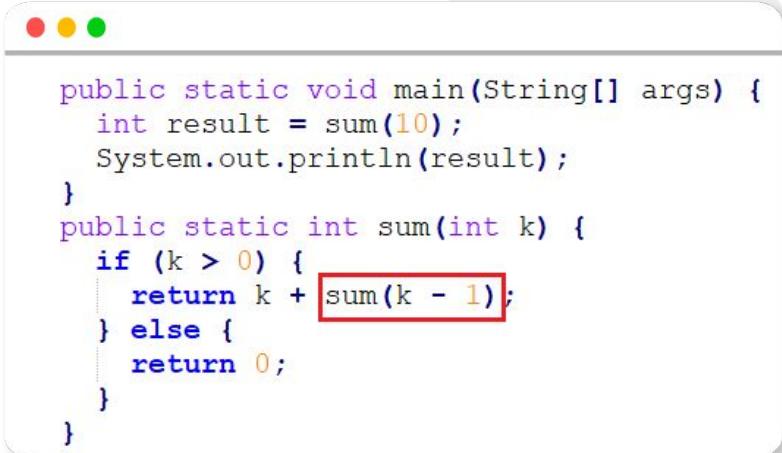
A screenshot of a Java code editor window. The code is as follows:

```
public static void main(String[] args) {
    int result = sum(10);
    System.out.println(result);
}
public static int sum(int k) {
    if (k > 0) {
        return k + sum(k - 1);
    } else {
        return 0;
    }
}
```

The line `sum(k - 1);` is highlighted with a red rectangular box, indicating it is the recursive call that needs to be enclosed in a code block to prevent an infinite loop.

Betul banget! Mirip kaya kita kalau lagi curhat sama diri sendiri di depan kaca, ya 😅

Terus, kalau  $k > 0$ , maka dia bakal memanggil diri sendiri, tapi kalau sebaliknya, maka method bakal return 0.



```
public static void main(String[] args) {
    int result = sum(10);
    System.out.println(result);
}
public static int sum(int k) {
    if (k > 0) {
        return k + sum(k - 1);
    } else {
        return 0;
    }
}
```

The image shows a screenshot of a Java code editor. It displays a simple recursive function named 'sum'. The function takes an integer 'k' as a parameter and returns an integer. If 'k' is greater than 0, it returns the value of 'k' plus the result of calling 'sum' with 'k-1'. If 'k' is not greater than 0 (i.e., it's 0 or less), it returns 0. In the code editor, the line 'return k + sum(k - 1);' is highlighted with a red rectangular box, indicating it is the part being discussed or executed.

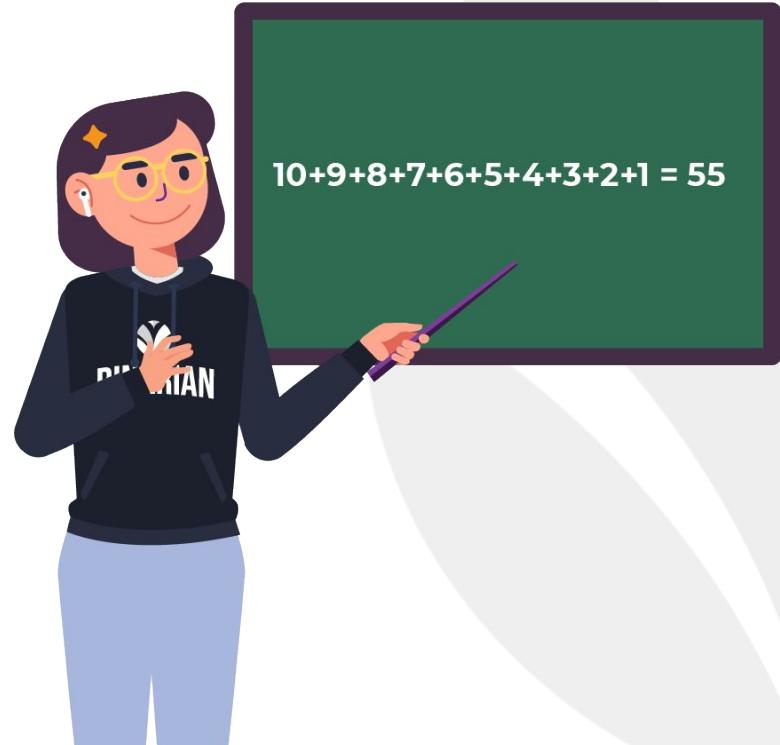
**Output dari contoh sebelumnya  
adalah 55~**

“Lho? kok bisa?”

Bisa, dong! Hal tersebut terjadi karena looping yang membentuk suatu penjumlahan kayak gini:

$$10+9+8+7+6+5+4+3+2+1 = 55$$

Wih, kayak lagi belajar matematika, ya 😅



### Biar makin paham nih, sekarang ayo kita berlatih!

1. Dengan menggunakan semua metode looping, buatlah pola-pola berikut!
  - 2 4 6 8 16 32
  - 2 4 8 32 216
2. Urutkan angka berikut dari besar ke kecil: 6 6 5 9 2 dengan menggunakan For!
3. Urutkan abjad berikut dari paling awal ke paling akhir M A K A N N A S I

Yukk, coba dicari tahu ya!



Keren nih kamu udah belajar banyak hal di topic ini! Materinya juga sudah mulai teknis dan udah mulai banyak ngodingnya.

Boleh cerita dong, materi mana sih yang challenging kamu pelajari pada topic 4 ini dan alasannya kenapa?



Nah, selesai sudah pembahasan kita di Chapter 1 Topic 4 ini.

Selanjutnya, kita bakal bahas tentang Working as A Backend Engineer.

Penasaran kayak gimana? Cus langsung ke topik selanjutnya~

