Struktur Kontrol

Struktur Kontrol Keputusan

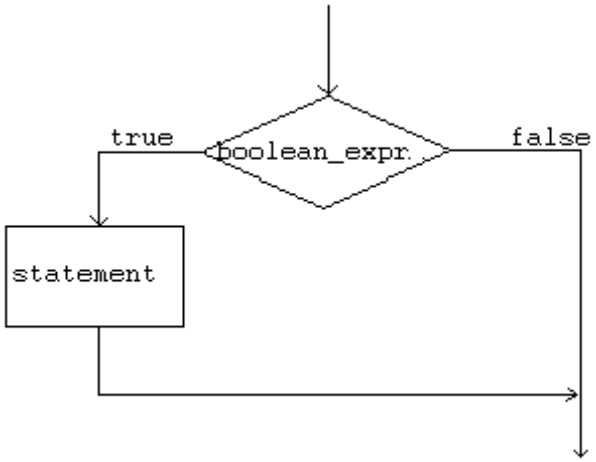
Struktur kontrol keputusan adalah statement dari Java yang mengijinkan user untuk memilih dan mengeksekusi blok kode dan mengabaikan blok kode yang lain.

1. *Statement if*

Statement-if menentukan sebuah statement (atau blok kode) yang akan dieksekusi jika dan hanya jika persyaratan boolean (*boolean statement*) bernilai *true.*

Bentuk dari statement if,

```
if( boolean_expression )
statement;
atau
if( boolean_expression ){
statement1;
statement2;
. . .  }
```



Gambar 1: Flowchart Statement If

dimana, boolean_expression adalah sebuah persyaratan *boolean* (*boolean statement*) atau *boolean* variabel.

Berikut ini adalah contoh code statement if,

```
int grade = 68;
if( grade > 60 ) System.out.println("Congratulations!");
atau
int grade = 68;
if( grade > 60 ){
System.out.println("Congratulations!");
System.out.println("You passed!");
}
```

Statement if-else

Statement if-else digunakan apabila kita ingin mengeksekusi sebuah statement dengan kondisi *true* dan statement yang lain dengan kondisi *false*.

Bentuk statement if-else,

```
if( boolean_expression )
statement;
else
statement;
```

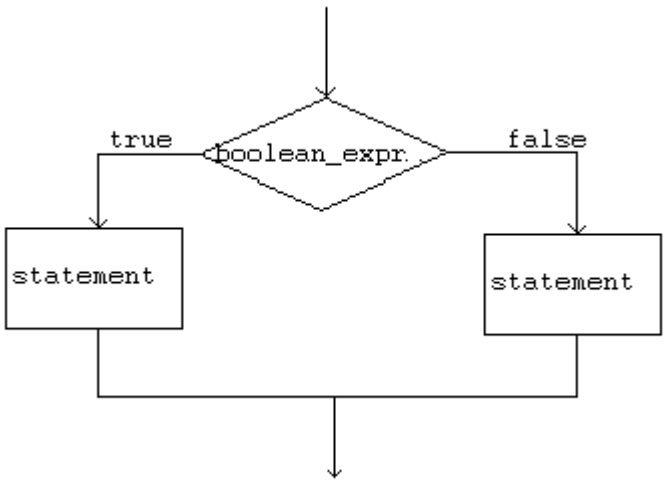
```
dapat juga ditulis seperti,
if( boolean_expression ){
statement1;
statement2;
. . .  }
else{
statement1;
statement2;
. . . }
```

Berikut ini contoh code statement if-else,

```
int grade = 68;
if( grade > 60 ) System.out.println("Congratulations!");
else System.out.println("Sorry you failed");
```

atau

```
int grade = 68;
if( grade > 60 ){
System.out.println("Congratulations!");
System.out.println("You passed!");
}
else{
System.out.println("Sorry you failed");
}
```



Gambar 2: Flowchart Statement If-Else

2. *Statement switch*

Cara lain untuk membuat percabangan adalah dengan menggunakan kata kunci **switch**. Dengan menggunakan switch kita bisa melakukan percabangan dengan persyaratan yang beragam.
Bentuk statement switch,

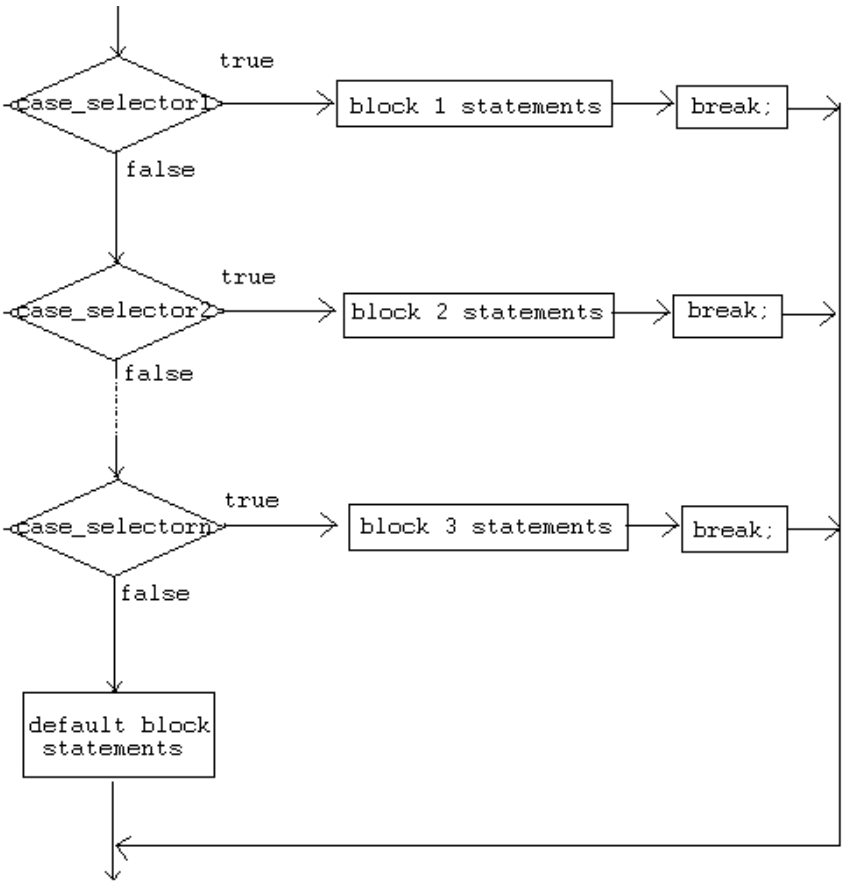
```
switch( switch_expression ){
case case_selector1:
statement1; //
statement2; //block 1
. . . //
break;
case case_selector2:
statement1; //
statement2; //block 2
. . . //
```

```
break;
. . .
default:
statement1; //
statement2; //block n
. . . //
break;
}
```

dimana, switch_expression adalah persyaratan **integer** atau **character** dan case_selector1, case_selector2 dan seterusnya adalah konstanta nilai integer yang unique (unik).

Ketika statement switch ditemukan, pertama kali Java memeriksa switch_expression, dan meloncat ke case dan mencocokkan nilai yang sama dengan persyaratannya. Program mengeksekusi statement dari awal sampai menemui statement break, dan melewati statement yang lain sampai akhir struktur switch.

Jika tidak ditemui case yang cocok, maka program akan mengeksekusi blok default. Bisa anda catat bahwa blok default adalah optional. Sebuah statement switch bisa tidak memiliki blok default.
CATATAN:
• Tidak seperti statement **if**, pada struktur switch statement dieksekusi tanpa memerlukan tanda kurung kurawal (**{}**).
• Ketika sebuah case pada statement switch menemui kecocokan, semua statement pada case tersebut akan dieksekusi. Tidak hanya demikian, statement lain yang berada pada case yang cocok juga dieksekusi.
• Untuk menghindari program mengeksekusi statement pada case berikutnya, kita menggunakan statement **break** sebagai statement akhir.



Gambar 4: Flowchart Statement Switch

Contoh statement switch

```
public class Grade
{
public static void main( String[] args )
{
  int grade = 92;
  switch(grade){
  case 100:
  System.out.println( "Excellent!" );
  break;
  case 90:
  System.out.println("Good job!" );
  break;
  case 80:
  System.out.println("Study harder!" );
  break;
  default:
  System.out.println("Sorry, you failed.");        }
}
}
```

Struktur Kontrol Perulangan

Struktur kontrol pengulangan adalah statement dari Java dimana kita bisa mengeksekusi blok code berulang-ulang dalam kurun nilai tertentu. Ada tiga macam jenis struktur kontrol pengulangan yaitu while, do-while, dan for-loops.

1. *while loop*

Statement while loop adalah statement atau blok statement yang diulang-ulang sampai mencapai kondisi yang cocok.
Bentuk statement while,

```
while( boolean_expression ){
statement1;
statement2;
. . . }
```

Statement di dalam while loop akan dieksekusi berulang-ulang selama *boolean_expression* bernilai true.
Contoh, pada code dibawah ini,

```
int i = 4;
while ( i > 0 ){
System.out.print(i);
i--; }
```

Contoh diatas akan mencetak angka 4321 pada layar. Perlu dicatat jika bagian i--; dihilangkan, akan menghasilkan looping yang tidak berhenti (**infinite loop**). Sehingga, ketika menggunakan while loop atau bentuk pengulangan yang lain, pastikan Anda memberikan statement yang membuat pengulangan berhenti pada suatu titik.
Berikut ini adalah beberapa contoh while loop,

Contoh 1:

```
int x = 0;
while (x<10) {
System.out.println(x);
x++; }
```

Contoh 2:

```
//infinite loop
while(true)
System.out.println(“hello”);
```

Contoh 3:
```
//no loops
// statement is not even executed
while (false)
System.out.println(“hello”);
```

2. *do-while loop*

Do-while loop mirip dengan while-loop. Statement di dalam do-while loop akan dieksekusi beberapa kali selama kondisi bernilai true.
Perbedaan antara while dan do-while loop adalah dimana statement di dalam do-while loop dieksekusi sedikitnya **satu kali.**
Bentuk statement do-while,

```
do{
statement1;
statement2;
. . .
}while( boolean_expression );
```

Statement di dalam do-while loop akan dieksekusi pertama kali, dan dilakukan pengecekan kondisi dari boolean_expression. Jika nilai tersebut belum mencapai nilai yang diinginkan, statement akan dieksekusi lagi.
Berikut ini beberapa contoh do-while loop:

Contoh 1:
```
int x = 0;
do
{
System.out.println(x);
x++;
}while (x<10);
```

Contoh ini akan memberikan output 0123456789 pada layar.

Contoh 2:
```
//infinite loop
do{
System.out.println(“hello”);
} while (true);
```

Contoh di atas akan melakukan pengulangan yang tidak berhenti untuk menulis “hello” pada layar.

Contoh 3:
```
//one loop
// statement is executed once
do
System.out.println(“hello”);
while (false);
```

Contoh di atas akan memberikan output hello pada layar.

3. *for loop*

Seperti pada struktur pengulangan sebelumnya yaitu melakukan pengulangan eksekusi code beberapa kali.
Bentuk dari for loop,

```
for (InitializationExpression; LoopCondition; StepExpression){
statement1;
statement2;
. . . }
```

dimana,
InitializationExpression – inisialisasi dari variabel loop.
LoopCondition - membandingkan variabel loop pada nilai batas.
StepExpression - melakukan update pada variabel loop.
Berikut ini adalah contoh dari for loop,

```
int i;
for( i = 0; i < 10; i++ ){
System.out.print(i);
}
```

Pada contoh ini, statement i=0 merupakan inisialisasi dari variabel. Selanjutnya, kondisi i<10 diperiksa. Jika kondisi bernilai true, statement di dalam for loop dieksekusi. Kemudian, statement i++ dieksekusi, dan dilakukan pengecekan kondisi. Kondisi ini akan dilakukan berulang-ulang sampai mencapai nilai yang salah (false).

Contoh tadi, adalah contoh yang sama dari while loop,

```
int i = 0;
while( i < 10 ){
System.out.print(i);
i++;
}
```

- ***break statement***

Statement break memiliki dua bentuk: unlabeled dan labeled.

  - **Unlabeled break statement**

    Unlabeled menghentikan jalannya statement switch. Anda bisa juga menggunakan bentuk unlabeled untuk menghentikan for, while atau do-while loop.
    Contohnya,

```
String names[] = {"Beah", "Bianca", "Lance", "Belle",
"Nico", "Yza", "Gem", "Ethan"};
String searchName = "Yza";
boolean foundName = false;
for( int i=0; i< names.length; i++ ){
if( names[i].equals( searchName )){
foundName = true;
break; }
}
if( foundName ){
System.out.println( searchName + " found!" );
}
else{
System.out.println( searchName + " not found." );
}
```

    Pada contoh ini, jika string “Yza” ditemukan, pengulangan pada for loop akan dihentikan dan akan melanjutkan ke proses berikutnya.

- **Labeled break statement**

Bentuk labeled form dari statement break akan menghentikan statement luar, dimana diidentifikasikan berupa label pada statement break. Program berikut ini akan mencari nilai dalam array dua dimensi. Terdapat dua pengulangan bersarang (nested loop). Ketika sebuah nilai ditemukan, labeled break akan menghentikan statement yang diberi label searchLabel, dimana label ini berada di luar.

```
int[][] numbers = {{1, 2, 3}, {4, 5, 6},
{7, 8, 9}};
int searchNum = 5;
boolean foundNum = false;
searchLabel:
for( int i=0; i<numbers.length; i++ ){
for( int j=0; j<numbers[i].length; j++ ){
if( searchNum == numbers[i][j] ){

foundNum = true; break searchLabel;
}
}
}
if( foundNum ){
System.out.println( searchNum + " found!" );
}
else{
System.out.println( searchNum + " not found!" );
}
```

Statement break menghentikan sementera labeled statement; ia tidak lagi menjalankan flow control pada label. Flow control pada label akan di-transfer secara otomatis mengikuti labeled statement.

Continue statement

Statement continue memiliki dua bentuk: unlabeled dan labeled. Anda dapat menggunakan statement continue untuk melewati pengulangan dari for, while, atau do-while loop yang sedang berjalan.

- **Unlabeled continue statement**

Bentuk unlabeled akan melewati akhir statement pada bagian yang dalam dan memeriksa boolean expression yang mengkontrol loop, pada dasarnya akan melewati bagian pengulangan pada loop.

Berikut ini adalah contoh dari penghitungan angka dari “Beah” dalam suatu array.

```
String names[] = {"Beah", "Bianca", "Lance", "Beah"};
int count = 0;
for( int i=0; i<names.length; i++ ){
if( !names[i].equals("Beah") ){
continue; //skip next statement
}
count++;
}
System.out.println("There are " + count + " Beahs in the list");
```

- **Labeled continue statement**

Bentuk labeled akan melanjutkan sebuah statement dengan melewati pengulangan yang sedang berjalan dari loop terluar yang diberi label (tanda).

```
outerLoop:
for( int i=0; i<5; i++ ){
for( int j=0; j<5; j++ ){
System.out.println("Inside for(j) loop"); //message1
if( j == 2 ) continue outerLoop;
}
System.out.println("Inside for(i) loop"); //message2
}
```

Pada contoh ini, pesan ke-2 tidak dicetak, karena statement continue akan melewati pengulangan yang sedang berjalan.

Return statement

Statement return digunakan untuk keluar dari sebuah fungsi (method). Statement return memiliki dua bentuk: menggunakan sebuah nilai, dan tidak memberikan nilai.

Untuk memberikan sebuah nilai, cukup berikan nilai (atau ekspresi yang menghasilkan sebuah nilai) sesudah return. Contohnya,

```
return ++count;
atau
return "Hello";
```

Tipe data dari nilai yang diberikan harus sama dengan tipe dari fungsi yang dideklarasikan. Ketika sebuah method void dideklariskan, gunakan bentuk return yang tidak memberikan nilai. Contohnya,

```
return;
```