STRUKTUR KONTROL

OBJEKTIF :

1. Mahasiswa Mampu Memahami Struktur Kontrol pada Java.
2. Mahasiswa Mampu Menggunakan *Software* IntelliJ IDEA dalam Pembuatan Perulangan dan Percabangan.

4.1 PERULANGAN

Perulangan atau ***looping*** merupakan sebuah metode untuk mengerjakan perintah yang berulang-ulang. Terdapat beberapa jenis perulangan pada Java, yaitu:

- `for`
- `while`
- `do-while`

Pernyataan-pernyataan itu menciptakan *loop*. *Loop* secara berulang mengeksekusi barisan instruksi yang sama sampai kondisi akhir ditemui.

4.1.1 PERULANGAN FOR

Perulangan `for` merupakan perulangan yang akan melakukan eksekusi perintah yang telah diketahui jumlah banyaknya. Sehingga perulangan `for` akan melakukan perulangan kode sejumlah tertentu. Perulangan ini terstruktur untuk mengulangi kode sampai tercapai batas yang telah ditentukan. Sintaks perulangan `for` adalah:

```
for (nilai_awal;kondisi;modifier)
Statement
```

Jika hanya satu pernyataan yang hendak diulang, maka diperbolehkan tidak memakai pasangan kurung kurawal. Perulangan `for` mengulangi *statement* sejumlah tertentu menggunakan:

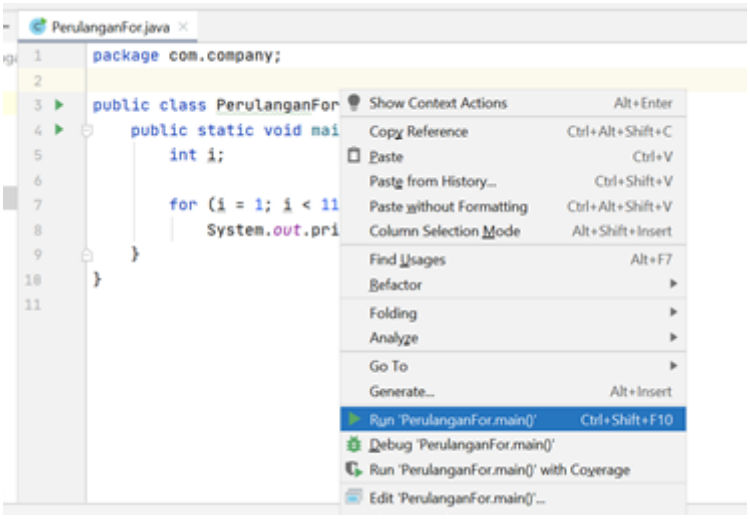
- `nilai_awal`, untuk deklarasi variabel kendali perulangan atau menginisialisasi nilai awal dimana menjadi titik awal perulangan dimulai.
- `kondisi`, membandingkan variabel kendali perulangan dengan nilai batas dengan memberikan kondisi tertentu.
- `modifier`, menspesifikasikan cara variabel kendali dimodifikasi sebelum iterasi berikutnya atau bagian untuk memberikan penambahan nilai atau pengurangan.

Berikut adalah contoh program perulangan dari 1 sampai 10:

```java
package com.integratedlaboratory.program;

public class PerulanganFor {
    public static void main(String args[]){
        int i;


        for (i = 1; i < 11; i++)
            System.out.println(i);
    }
}
```

Perintah:

Tekan tombol Ctrl+Shift+F10 pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada file java seperti berikut:



Hasil program:

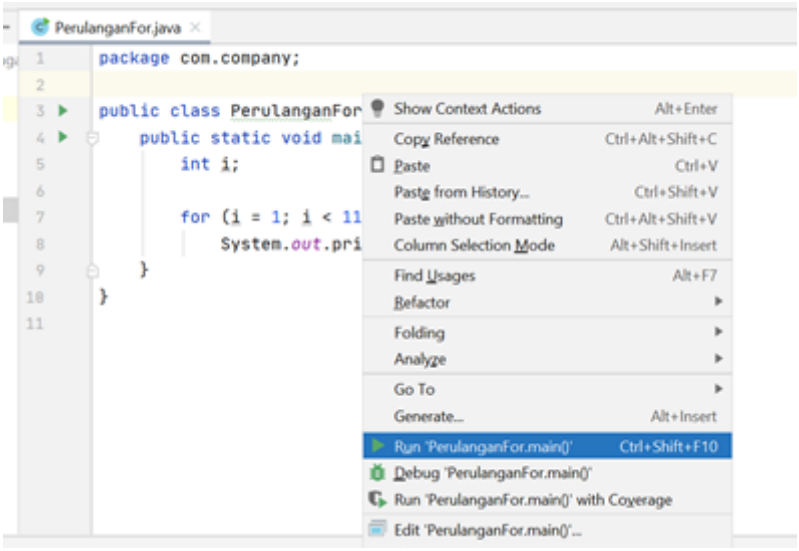
4.1.1.1 Deklarasi Variabel Kendali *loop* di dalam for

Looping pada bahasa pemrograman Java adalah fitur yang memfasilitasi eksekusi program dari sekelompok instruksi atau fungsi secara berulang ketika beberapa kondisi tertentu bernilai benar. `for` menyediakan solusi ringkas dari struktur *looping* atau perulangan (iterasi) dan mudah melakukan *debug* pada struktur *looping*.

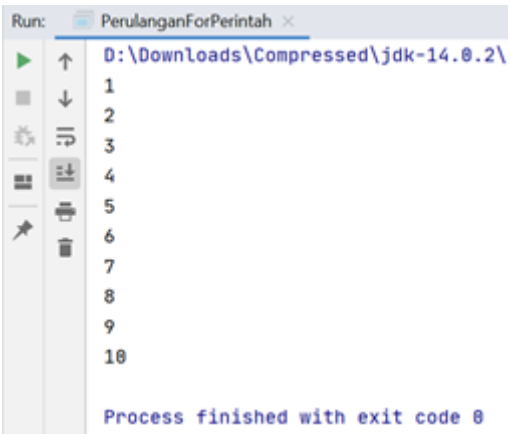
Dalam melakukan `for`, variabel yang digunakan untuk mengendalikan *loop* tersebut dideklarasikan di dalam inisialisasi `for` .

```java
package com.integratedlaboratory.program;

public class PerulanganForPerintah {
    public static void main(String args[]){
        for (int i = 1; i < 11; i++)
            System.out.println(i);
    }
}
```

Tekan tombol Ctrl+Shift+F10 pada IntelliJ IDEA atau dengan melakukan klik kanan pada file java seperti berikut:



Hasil program:



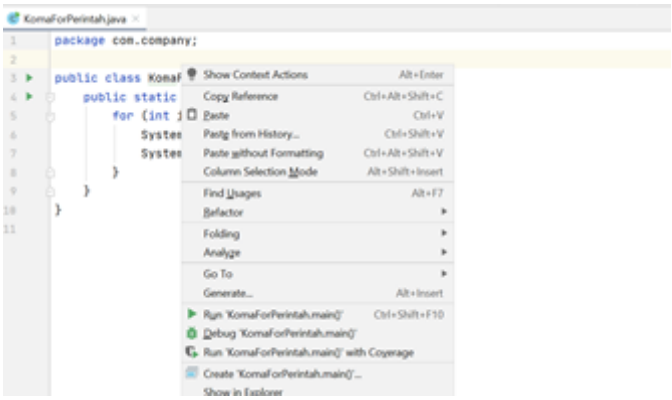
Ketika mendeklarasikan variabel di dalam `for`, maka harus diperhatikan bahwa lingkup variabel berakhir ketika `for` selesai dieksekusi. Maka dari itu, pastikan bahwa lingkup variabel yang diberikan terbatas di dalam `for` dan variabel tidak berada di luar `for`. Jika memerlukan variabel kendali untuk *loop*, maka deklarasikan di luar `for`. Ketika variabel kendali `for` tidak diperlukan, maka deklarasikan di dalam `for`.

4.1.1.2 Penggunaan Koma dalam For

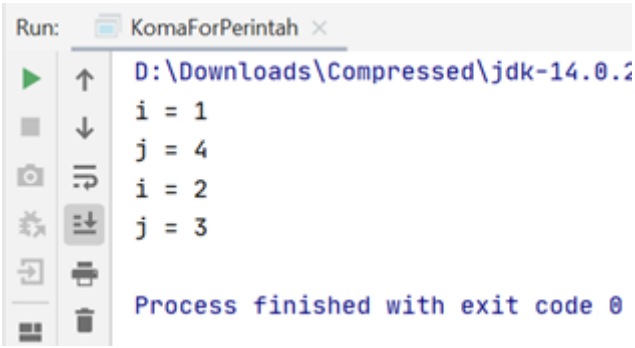
Pemberian pernyataan untuk inisialisasi dan iterasi di `for`, seringkali diberikan lebih dari satu pernyataan. Java mengizinkan satu variabel atau lebih berada di dalam pernyataan *loop*. Namun, masing-masing pernyataan tersebut harus dipisahkan dengan tanda koma (seperti pada baris ke-5). Contoh seperti berikut:

```java
package com.integratedlaboratory.program;

public class KomaForPerintah {
    public static void main(String args[]){
        for (int i=1, j=4; i<j; i++, j--) {
            System.out.println("i = " + i);
            System.out.println("j = " + j);
        }
    }
}
```

Tekan tombol Ctrl+Shift+F10 pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada file java seperti berikut:



Hasil program:



Pada contoh di atas, bagian inisialisasi berisi deklarasi dan pemberian nilai untuk i dan j. Dua pernyataan tersebut dipisahkan dengan koma dan dieksekusi tiap kali *loop* diulangi.

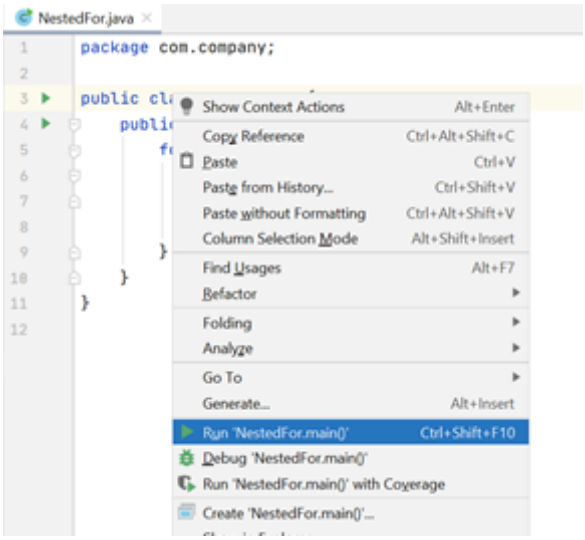
4.1.1.3 Pernyataan For Bersarang

Java juga mengizinkan adanya *loop* disarangkan di *loop* lain atau satu *loop* berada pada *loop* lain. Penggunaan `for` di dalam sebuah for diperbolehkan dalam Java. *Loop* bersarang akan mengeksekusi *loop* yang berada di dalam terlebih dahulu hingga selesai, baru menjalankan *loop* yang berada di luar. Seperti contoh berikut, `for` yang berada di dalam (`variabel j`) akan dijalankan terlebih dahulu hingga ditemukan kondisi yang sesuai, setelah itu `for` yang berada di luar (`variabel i`) akan dijalankan.

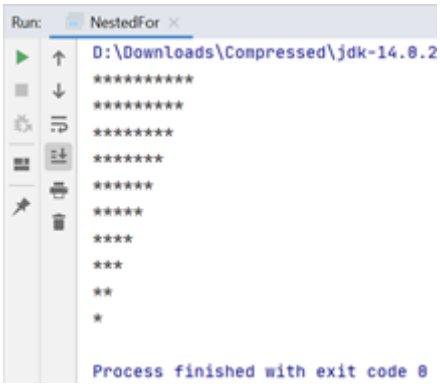
Contoh:

```
package com.integratedlaboratory.program;

public class NestedFor {
    public static void main(String args[]){
        for (int i=0; i<10;i++) {
            for (int j=i; j<10; j++) {
                System.out.print("*");}
            System.out.println();
        }
    }
}
```

Tekan tombol Ctrl+Shift+F10 pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada file java seperti berikut:



Hasil program:



4.1.2 PERULANGAN WHILE

Perulangan `while` digunakan untuk mengulang suatu perintah perulangan yang belum diketahui jumlahnya. Perulangan `while` akan terus dijalankan selama kondisi yang dieksekusi bernilai *true*.

Sintaks perulangan `while` adalah:

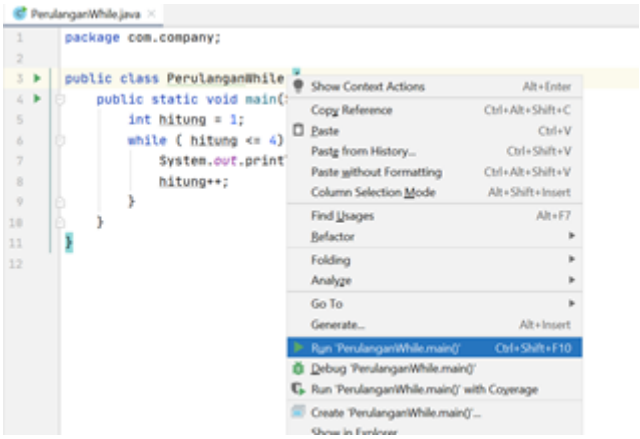
```
while (kondisi)
    Statement
```

Jika kondisi bernilai *true*, maka `Statement` akan terus dieksekusi dan proses akan berlanjut diulangi. Penting diketahui bahwa, kondisi berada sebelum badan pernyataan. Sehingga ketika kondisi sejak awal bernilai *false*, maka `Statement` tidak akan pernah dieksekusi. Hal ini merupakan perbedaan penting antara perulangan `while` dan `do-while`.

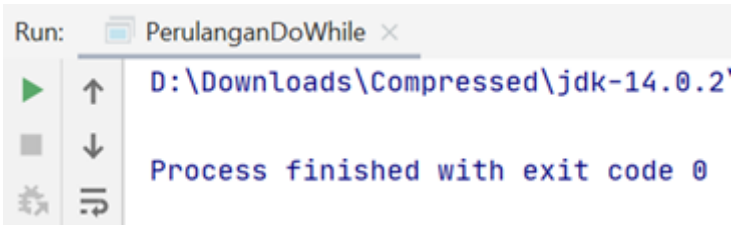
Berikut ini merupakan contoh perulangan `while`, dimana kondisi sejak awal sudah bernilai *false*. Sehingga tidak ada `statement` yang dijalankan pada program.

```
package com.integratedlaboratory.program;

public class PerulanganWhile {
    public static void main(String[] args) {
        int i = 6;
        while ( i <= 5) {
            System.out.println("perulangan ke-" + i);
            i++;
        }
    }
}
```

Tekan tombol Ctrl+Shift+F10 pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada file java seperti berikut:



Hasil Program:



4.1.3 PERULANGAN DO-WHILE

Perulangan `do-while` hampir sama dengan perulangan `while`, namun pada perulangan `do-while` diberikan pernyataan terlebih dahulu, baru kemudian dilakukan pengecekan kondisi. Maka pengecekan kondisi dilakukan di akhir blok `statement`. Perulangan akan dilakukan jika kondisi yang didefinisikan bernilai *true*. Namun, jika kondisi tidak terpenuhi, maka proses perulangan akan dilakukan **minimal satu kali**.

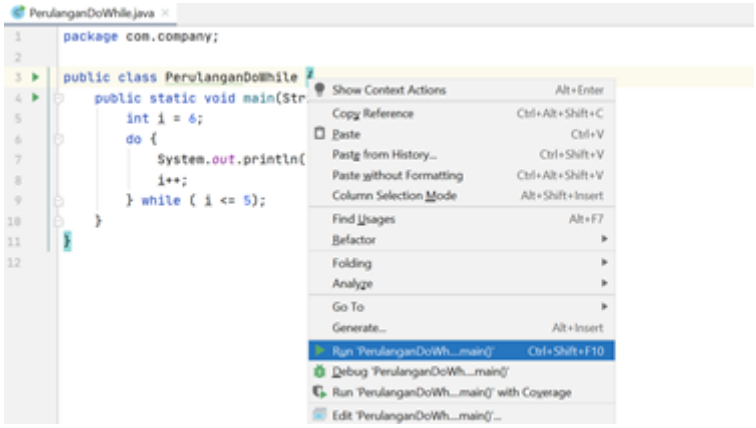
Berikut ini sintaks untuk perulangan `do-while`:

```
do
    Statement
while (kondisi);
```

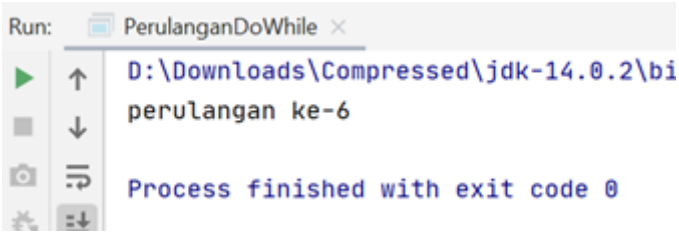
Contoh:

```
package com.integratedlaboratory.program;

public class PerulanganDoWhile {
    public static void main(String[] args) {
        int i = 6;
        do {
            System.out.println("perulangan ke-" + i);
            i++;
        } while ( i <= 5);
    }
}
```

Tekan tombol Ctrl+Shift+F10 pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada file java seperti berikut:



Hasil program:



Pada contoh di atas, diberikan kondisi yang bernilai *false* seperti contoh pada perulangan `while`. Namun terlihat pada contoh perulangan `do-while`, walaupun kondisi bernilai *false*, `statement` tetap dijalankan sebanyak satu kali.

4.2 PERCABANGAN

Percabangan merupakan beberapa pilihan *statement*(pernyataan) yang akan di eksekusi berdasarkan kondisi tertentu yang diberikan. Percabangan merupakan alur program yang bercabang. Percabangan pada Java digunakan untuk memilih dan mengeksekusi blok kode spesifik, serta mengabaikan blok kode lainnya. Terdapat beberapa jenis percabangan pada Java, yaitu:

- Pernyataan `if`
- Pernyataan `if-else`
- Pernyataan `switch`

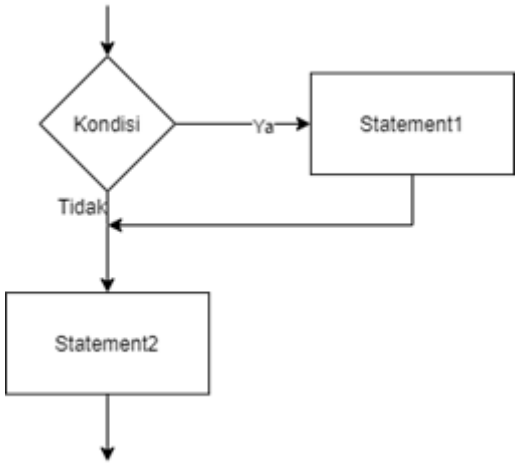
Dengan percabangan, kita dapat mengendalikan alur eksekusi program berdasarkan kondisi yang diberikan.

4.2.1 PERNYATAAN IF

Pernyataan `if` pada Java, menyatakan pernyataan akan dieksekusi bila memenuhi syarat atau kondisi tertentu. Pernyataan `if` hanya memiliki sebuah pilihan saja. Pernyataan `if` hanya akan mengeksekusi pilihan jika kondisi bernilai *true*. Jika kondisi bernilai *false*, maka tidak akan terjadi apa-apa dan lanjut mengeksekusi ke perintah berikutnya. Sintaks pernyataan `if` ini sebagai berikut:

```
if (kondisi) {
    Statement1;
    }
Statement2;
```

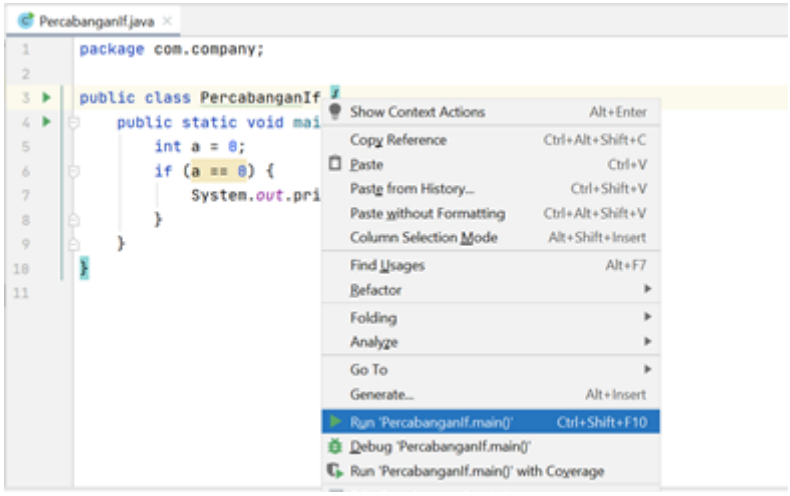
Alur dari pernyataan `if` digambarkan sebagai berikut:



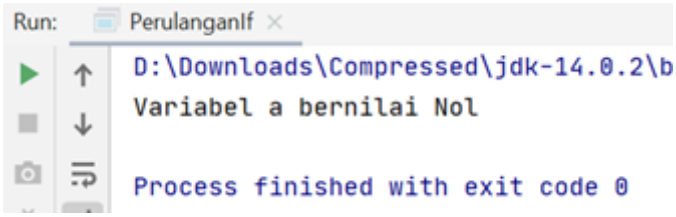
Dari gambaran di atas, jika kondisi bernilai benar maka akan dijalankan `statement1` yang terdapat di dalam tanda kurung kurawal. Dan jika kondisi tidak sesuai, maka program akan dijalankan mengikuti `statement2` atau perintah lain yang diberikan di luar pernyataan `if`. Berikut ini merupakan contoh program dengan menggunakan pernyataan `if`:

```
package com.integratedlaboratory.program;

public class PercabanganIf {
    public static void main(String args[]){
        int a = 0;
        if (a == 0) {
            System.out.println("Variabel a bernilai Nol");
        }
    }
}
```

Tekan tombol Ctrl+Shift+F10 pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada file java seperti berikut:



Hasil program:

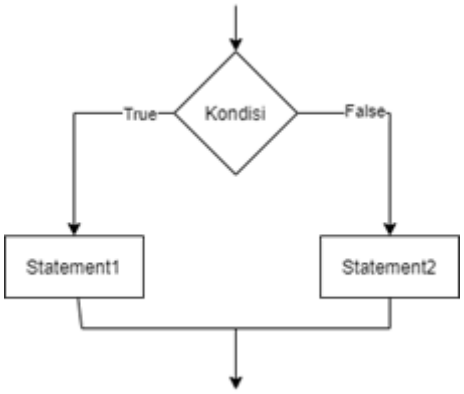


4.2.2 PERNYATAAN IF-ELSE

Pernyataan `if-else` terdiri dari beberapa pernyataan dengan kondisi *true* dan pernyataan lain sebagai alternatif jika kondisi *false*. Pernyataan `if-else`, dapat memilih salah satu dari dua kemungkinan kemunculan. Berikut ini merupakan sintaks dari pernyataan `if-else`:

```
if (kondisi) {
    Statement1
} else {
    Statement2
}
```

Alur dari pernyataan `if-else` digambarkan sebagai berikut:



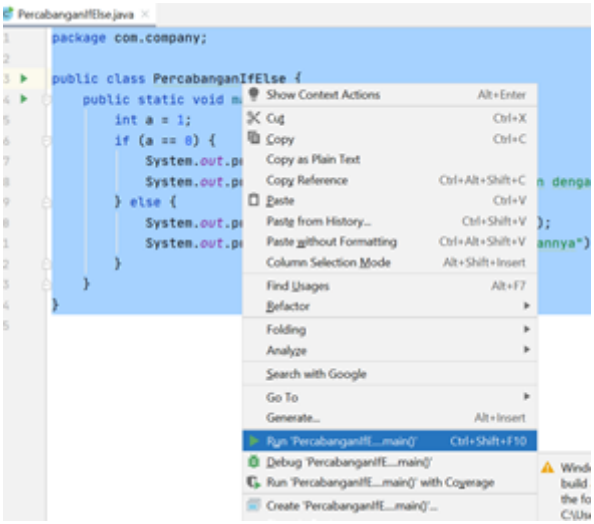
Dari gambaran di atas, jika kondisi ketika dievaluasi bernilai *true*, maka `statement1` akan dieksekusi. Jika kondisi ketika dievaluasi bernilai *false*, maka `statement2` yang akan dieksekusi. Berikut ini merupakan contoh program menggunakan pernyataan `if-else`:

```
package com.integratedlaboratory.program;

public class PercabanganIfElse {
    public static void main(String args[]){
```

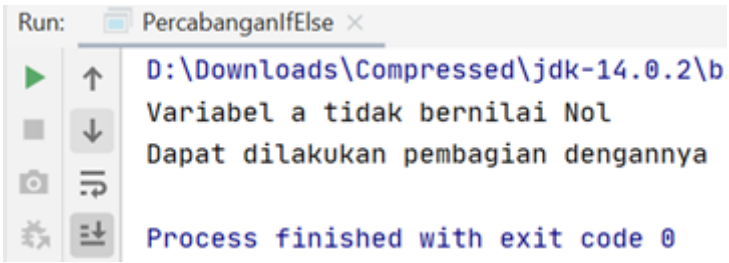
```
        int a = 1;
        if (a == 0) {
            System.out.println("Variabel a bernilai Nol");
            System.out.println("Tidak dapat dilakukan pembagian dengannya");
        } else {
            System.out.println("Variabel a tidak bernilai Nol");
            System.out.println("Dapat dilakukan pembagian dengannya");
        }
    }
}
```

Perintah:

Tekan tombol Ctrl+Shift+F10 pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada file java seperti berikut:



Hasil program:



4.2.3 PERNYATAAN SWITCH

Pernyataan `switch` digunakan untuk beberapa perbandingan kondisi yang berbeda. Pernyataan `switch` dimaksudkan untuk menangani banyak kemungkinan kemunculan. Pernyataan `switch` memiliki beberapa kondisi pada setiap pernyataannya, program akan dieksekusi atau dijalankan hingga ditemukan kondisi yang cocok dengan pernyataan `switch` yang diberikan. Sintaks dari pernyataan `switch` adalah:

```
Switch (variabel) {
    case nilaikonstan1:
        StatementList1
case nilaikonstan2:
        StatementList2
default:
        defaultStatementList
}
```

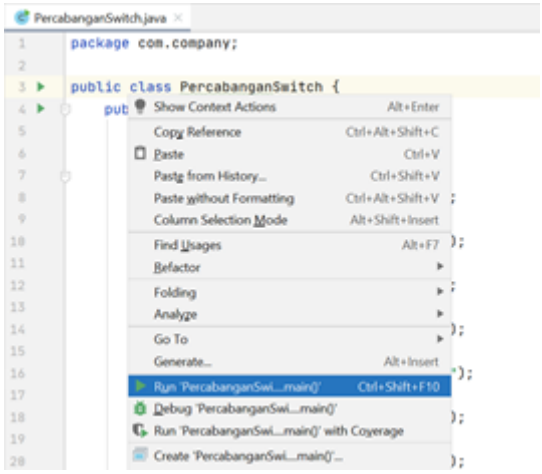
Alur dari pernyataan `switch` adalah:

Dari gambar di atas, ketika kondisi `switch` terpenuhi, maka akan dilakukan evaluasi apakah perintah yang masuk memenuhi `case` dengan `nilaikonstan1` atau `nilaikonstan2`. Setelah ditemukan pilihan `case` yang tepat, maka akan dijalankan sesuai *statement* yang diberikan pada `case` tersebut. Jika tidak menemukan `case` yang tepat, maka akan dijalankan `defaultStatementList`. Berikuti ini merupakan contoh program menggunakan pernyataan `switch`:

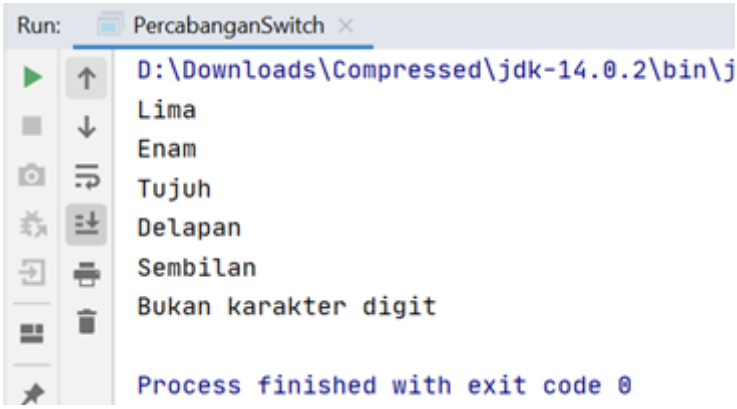
```
package com.integratedlaboratory.program;

public class PercabanganSwitch {
    public static void main (String args[]) {
        int a;
        a = 5;
        switch (a) {
            case 0: System.out.println("Nol");
            case 1: System.out.println("Satu");
            case 2: System.out.println("Dua");
            case 3: System.out.println("Tiga");
            case 4: System.out.println("Empat");
            case 5: System.out.println("Lima");
            case 6: System.out.println("Enam");
            case 7: System.out.println("Tujuh");
            case 8: System.out.println("Delapan");
            case 9: System.out.println("Sembilan");
            default: System.out.println("Bukan karakter digit");
        }
    }
}
```

Perintah:

Tekan tombol Ctrl+Shift+F10 pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada file java seperti berikut:



Hasil program:



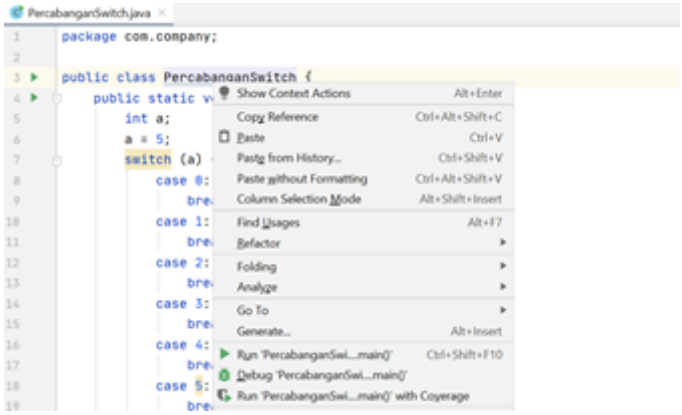
Pernyataan `break` adalah opsional. Jika kita meniadakan `break`, maka eksekusi akan terus dilakukan ke pernyataan `case` berikutnya. Pernyataan `break` diadakan sebagai batas akhir dari `statement`.

```
package com.integratedlaboratory.program;

public class PercabanganSwitch {
    public static void main (String args[]) {
        int a;
        a = 5;
        switch (a) {
            case 0: System.out.println("Nol");
                break;
            case 1: System.out.println("Satu");
                break;
            case 2: System.out.println("Dua");
                break;
            case 3: System.out.println("Tiga");
                break;
            case 4: System.out.println("Empat");
                break;
            case 5: System.out.println("Lima");
                break;
            case 6: System.out.println("Enam");
                break;
            case 7: System.out.println("Tujuh");
                break;
            case 8: System.out.println("Delapan");
                break;
            case 9: System.out.println("Sembilan");
```

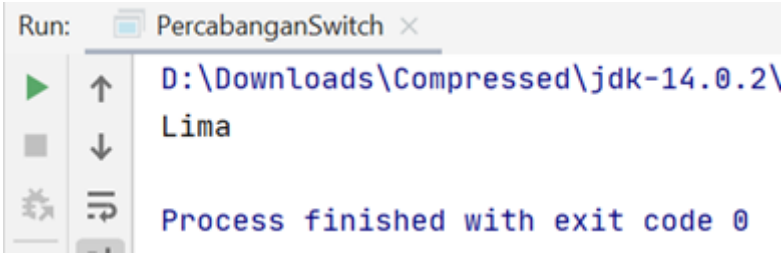
```
                break;
            default: System.out.println("Bukan karakter digit");
        }
    }
}
```

Perintah:

Tekan tombol Ctrl+Shift+F10 pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada file java seperti berikut:



Hasil program:



Berikut adalah tiga fitur penting dalam pernyataan `switch`, yaitu:

1. Pernyataan `switch` berbeda dengan pernyataan `if`, dimana switch hanya dapat menguji kesamaan, sedangkan pernyataan `if` dapat melakukan evaluasi sembarang tipe ekspresi `boolean`. Dengan demikian, pernyataan `switch` hanya akan mencocokkan di antara nilai-nilai ekspresi dan konstanta-konstanta `case`.
2. Tidak ada konstanta `case` di dalam blok `case` dengan nilai yang identik sama. Namun pernyataan `switch` (`nested switch`) dengan `switch` lain dapat memiliki konstanta yang sama.
3. Pernyataan `switch` biasanya lebih efisien dibanding `if` yang bersarang di dalam.

REFERENSI:

[1] Hariyanto, Bambang. 2017. *Esensi-Esensi Bahasa Pemrograman Java Revisi Kelima*. Bandung: Informatika.

[2] Muhardian, Ahmad. 2016. "Belajar Java: Memahami 2 Jenis Perulangan dalam Java", https://www.petanikode.com/java-perulangan/. Diakses pada 20 Juli 2020.

[3] Muhardian, Ahmad. 2015. "Belajar Java: Memahami 3 Bentuk Percabangan dalam Java", https://www.petanikode.com/java-percabangan/. Diakses pada 20 Juli 2020.